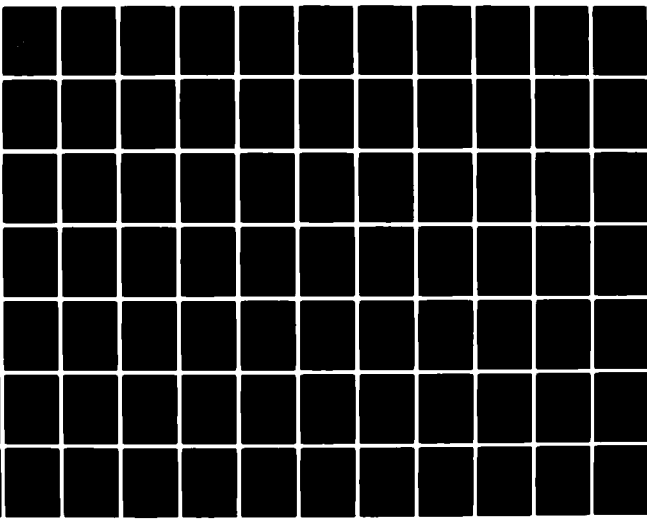


AD-A118 468

CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/6 9/2
PRODUCTION SYSTEMS AS A PROGRAMMING LANGUAGE FOR ARTIFICIAL INT--ETC(U)
DEC 76 M D RYCHENER: F44620-73-C-0074
NL

UNCLASSIFIED

1 of 2
of 2 pages





DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DTIC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

**Production Systems as a Programming Language
for Artificial Intelligence Applications**

Volume II

Michael D. Rychener

December 1976

**Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa 15213**

This report reproduces a dissertation submitted to the Department of Computer Science at Carnegie-Mellon University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

Table of Contents

For Volume II

Chapter	Page
III. A Production System Implementation of EPAM	III-1
Start of Appendices	III-27
End of Appendices	III-40
IV. GPSR: A Production System Implementation of GPS	IV-1
Start of Appendices	IV-79
End of Appendices	IV-108

Preface to Volume II

This volume contains two chapters, covering work with production systems in the areas of self-augmenting systems and problem solving using heuristic search. Chapter III describes a simple verbal learning system, which builds a discrimination network of productions. Chapter IV describes an implementation of the classic means-ends problem-solving system, GPS. Each chapter has an abstract and a detailed table of contents. It is assumed that the reader has some familiarity with Volume I of this report, which discusses the goals and conclusions of the thesis as a whole, and which introduces the production system language in which the systems in this volume are implemented. The chapters have a similar organization, starting with a general description of the task performed by the system, and proceeding to a description of the system and its behavior. There are sections that discuss issues with respect to the task itself and with respect to the use of production systems.



A-23

Chapter III

A Production System Implementation of EPAM

✓
Abstract. EPAM is a simple model of verbal learning that was developed to simulate certain features of human learning, but it has also turned out to be useful for certain kinds of discriminations in AI programs. This chapter describes a production system for EPAM, featuring the automatic addition of productions by the basic system to represent incremental learning of three-letter nonsense syllables. The design of the network represented by the added productions is discussed and its growth described. Details of the EPAM production system raise several issues with respect to general EPAM variations and with respect to production system issues such as the right set of production-building primitives. A comparison of the present program to a similar one by Waterman, using a radically different production system architecture, is carried out, highlighting the advantages of the present one.

7

2 2

EPAM

Table of Contents

For Chapter III

SECTION	PAGE
A Introduction	III-1
B Alternatives in the Design of EPAM Net Productions	III-3
C An Example of the Building of Net Productions	III-5
D Details of EPAM	III-9
D.1 How the program works	III-9
Fig. D.1 Abstract productions for EPAM	III-10
D.2 Meanings of EPAM predicates	III-13
E Some Issues Raised by EPAM	III-15
E.1 Production systems issues	III-15
E.2 EPAM issues	III-16
F A Comparison to Waterman's EPAM2	III-19
Fig. F.1 A comparison of behaviors on a seven-pair test	III-20
Fig. F.2 Networks produced by the two programs	III-21
Fig. F.3 A comparison by functional units	III-23
G Conclusions	III-25
G.1 Summary of conclusions	III-25
G.2 Further research	III-25
G.3 Acknowledgement	III-26
G.4 References	III-26
APPENDIX	PAGE
A Program Listing	III-28
B Cross-Reference of EPAM Predicates	III-29
C Detailed Trace on Three-Pair Test	III-30
D Summary of Behavior for Other Tests	III-38
E Waterman's EPAM2	III-40

EPAM

A. Introduction

This chapter describes a production system (PS)[•] implementation of Feigenbaum's EPAM (1963), an elementary perceiving and memorizing system designed to simulate the learning of simple nonsense-syllable associations. Each association is a pair of three-letter syllables, called the stimulus and response. The association is mediated internally by memory cues. As pairs are learned a discrimination network is built up to recognize stimuli, producing cues, and to recognize cues, producing responses. The following is an example of pairs to be learned:

Stimulus	Response
PUK	RIN
KOF	LYB
POM	LUB

The way this works experimentally is that the experimenter gives a stimulus syllable and then a response syllable. On the first pass through the list of pairs, the subject can make no correct responses, but on later passes, he tries to produce the response immediately after the stimulus is given. Producing a response is apparently achieved by creating first a discrimination on the stimulus, to produce an internal memory cue. The memory cue then leads by association to data sufficient to produce a response. The only feedback the subject receives is the correct response. Pairs like the ones above have pitfalls designed into them. For instance, the syllables "PUK" and "POM" have initial letters the same, and there is some chance that a subject will confuse them, since the theory says that a subject will not remember a complete syllable, but rather will be able to discriminate only on the basis of something like "P--", P followed by something unknown or indistinct. Thus a second pass through the syllables will be required, to extend that partial discrimination to something like "P-K" or "P-M". (Psychological evidence dictates against the variants "PU-" or "PO-", since apparently the outer positions are more easily made use of.) Similarly, memory cues that are produced as a result of discriminating stimuli are partial, for instance "R--" or "L--" for the above task. The "R--" cue will suffice to uniquely determine a response, since no other response syllable starts with R. But the "L--" cue is ambiguous, and in fact if cues are extended by one letter each time the subject is exposed to a pair, two more exposures will be required, to extend the cues to "L-B" and then "LUB" and "LYB".

The approach here encodes the net as a set of Ps, but otherwise follows closely the original mechanisms of discrimination and learning. This is an initial attempt to explore the mechanisms needed for a PS to add to itself. The task, however, lacks much of what is interesting about the process of learning, as is indicated by the following attributes: (1) learning takes place at well-defined times; (2) the Ps on which learning takes place are of a rigid format, and are modified in specific fixed ways; (3) the credit-assignment problem is easy, since the fault may lie in only one of two Ps, the one that discriminates the stimulus, or the one that discriminates the cue to produce a response.

[•] PS will be used to abbreviate production system, plural PSs; P, to abbreviate production, plural Ps.

Nevertheless, much care was needed in the design of the program, with several discoveries along the way that required reworking of a large part of the program design. These discoveries mainly centered on the program's corrective action as a result of wrong reply. The initial design and an immediate successor failed to take into account all the possible combinations of correct and incorrect net Ps that played a part in producing the reply. (This comment may become clearer after details of the program are presented.) Once the details had been worked out more carefully, the program was easily completed to a satisfactory form. This illustrates the additional difficulties that can be encountered in designing a program indirectly, by designing a program that produces the program. Also, the simple description of EPAM in Feigenbaum (1963) fails to touch on many of the issues of design that led to the mentioned discoveries.

As background for this description, only the paper by Feigenbaum (1963) is necessary. There has been no attempt to compare the learning behavior of this program with any data from human subjects, although in cases where there is slight departure from the original EPAM design as portrayed in that paper, such a comparison might be useful. Throughout, EPAM will refer to the present PS version, unless specifically noted.

Section B discusses design alternatives for the representation of discrimination networks in PSs. Section C explains the workings of the network built by EPAM, avoiding details of EPAM itself, whose exposition is the subject of Section D. Section E discusses general issues with respect to both PSs and EPAM-like processes. Section F is a comparison to Waterman's EPAM2, which is implemented in a contrasting PS language. Section G summarizes, and points out problems for further investigation.

B. Alternatives in the Design of EPAM Net Productions

EPAM constructs a network of tests that is used to discriminate stimuli in order to produce the response half of a stimulus-response pair. This network is minimal in a global sense: it doesn't build up structures that recognize full syllables, as would be feasible in a computer program. Rather, it tries to base the network on a small amount of information, usually building up tests a letter at a time. Actually it does slightly more, adding a negative test for each positive one, regardless of whether some current task demand can make use of the negative test. The result is that sometimes the program can know that it doesn't know something, because it is making use of a negative test that hasn't been used before. The program builds the network gradually, with successive passes over the set of pairs bringing to light the necessary discriminations.

There are several ways to design the system, varying according to the way the net is represented:

- a. Simply using PSs as one would any other language, with the net represented as a data structure with an interpreter.
- b. Code each test in the net as a P; a net memory cycle would involve perhaps several P firings; some means of inter-communication between the Ps would be necessary.
- c. Code each association path through the net as an LHS, with each RHS representing the information stored at terminals of the net.

For the present implementation, alternative c. was chosen, for the following reasons. With respect to alternative a. the other two alternatives are better, because the focus of the study was to explore mechanisms for adding new Ps to an existing set. Alternative c. was judged to be better than b. because it corresponds to an efficient way to compile LHSs for matching. Such a net representation would be optimal in preventing duplication of tests on Working Memory. In a system that compiled Ps in this way, EPAM would result in a network in form as well as in intent. Hayes-Roth and Mostow (1975) have compiled Ps for speech understanding into such a network.

An additional design consideration for alternatives b. and c. is how to keep track of the information encoded in the network Ps, which is necessary when additions are to be made to the network.

- i. Selected information could be made available at all times in Working Memory.
- ii. The same information could be distributed to the RHS of each P, having it available only when the P fires, and erasing it after use (to simulate fading of short-term memory in humans). This follows the principle that long-term information is stored as Ps.
- iii. The system could be limited to knowing about a P only indirectly through its effects.

The first two alternatives essentially allow Ps to be inspected and their contents modified (given the appropriate PS operators), while the last might limit action on Ps to simply adding more Ps; for instance, Ps might be added to modify the effects of an undesirable P after they had happened, but before they had been converted to irreversible external

behavior or internal actions. The present approach is a compromise between the first two: information on a P is kept always in Working Memory, but is used only when the name of a P is available due to a recent firing of that P. The importance of this consideration was not realized until after EPAM was finished, or a cleaner (pure ii.) approach would have been used. In any case, the third alternative seems more difficult and is best left to another time.

C. An Example of the Building of Net Productions

This section will go through in detail the building of a simple piece of the network, emphasizing the network built rather than the building PS process. First, some general comments are necessary. The network is used to discriminate two types of inputs: stimuli from the outside, and memory cues stored internally in RHSs of Ps. When a stimulus is presented, a cue may be produced, which in turn is fed into the net, to possibly result in the assertion of an image which then becomes a reply. (Internally in this program, an image and a reply are identical, but other models have used different representations, so the naming distinction is retained here.) An image may be produced in the first firing, and a cue in the second, but these are erased appropriately. The stimuli are perceived in entirety, that is, all three letters of the syllable are available. Each letter is available separately, associated with a number indicating its position in the syllable. It may be that not all three letters will be used in a test, however. Cues are partial, being built up only as is necessary to produce the proper reply image. That is, a cue consists of one to three letters with associated positional information. Images are complete, leading to an entire syllable as a reply, although the wrong one may be produced. No internal information on an image is used - it is a symbol with neither positional structure nor examinable components. Strings composed of letters are taken to be perceptual-motor primitives, avoiding most details of modelling translation between modalities of encoding.

A network for a particular set of pairs is learned in several passes over the set. After each pair, the net is guaranteed to be correct for that pair but discriminations made for it may have obscured, or interfered with, previous discriminations, which were not as detailed. On the first pass, discriminations on the first letter are made. On the second, those are refined where necessary by adding tests on a second letter, etc. Within a pass, tests using the same letter in the same position, but in different syllables, may result in partially going beyond these bounds, because the discriminations extended at one point may be extended again on encountering similar ones. That is, discriminations are not made on a "second pass, second letter" basis, but are made flexibly according to specific task demands. Since the domain here is restricted to three-letter syllables, at most three passes are required. In accord with the original EPAM, letters are processed in an order that is not left-to-right, namely, the first letter, then the third, then the second. This 1-3-2 order will be referred to as the noticing order of syllable letters.

Each P in the net consists of tests on letters of the stimulus or cue, with a standard-format RHS consisting of optional cue, optional image, and a "fired" signal. A typical net P from the detailed example below is:

```
PH-1; "NET" :: LET1(V1) & R(V1)
=> EXISTS(C1) & FIRED('PH-1) & IMAGE('DUNNO) & LET1(C1) & P(C1)
& CUETRIPLE('PH-1,C1,'XX,'XX);
```

This means: "if letter-1 is R, then emit cue P and image DUNNO." Note that the LHS test is in two parts, testing the position of a letter (LET1) and what it is equal to (R). The RHS has four things representing the cue: EXISTS, which creates a new internal token; LET1, which is positional information for the cue letter; P for the letter's value; and CUETRIPLE,

holding information about the cue that is used in matching it, later in the process. Details on representation and other aspects are in Section D.

The following is an example of how a set of Ps representing a network will be displayed in this chapter. This net is the result of learning the three pairs FIB/NUM, RAM/POR, and PEK/NAM:

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
1:	F	R	P	N			else
	=>NUM	=>P--	=>NAM				=>---
	I ?	I ?	I POR				I NUM
2:				M		S	
						=>---	
						I NAM	
3:				A	S		
				=>---	=>---		
				I NAM	I NUM		

The Ps in the net are represented as columns, with rows representing LHS tests or RHS actions. The P PM-1 above is column (b). The meanings of the columns are as follows:

- (a) letter-1 F, cue NUM, image DUNNO.
- (b) letter-1 R, cue P, image DUNNO.
- (c) letter-1 p, cue NAM, image POR.
- (d) letter-1 N, letter-2 M, letter-3 A, cue null, image NAM.
- (e) letter-1 N, letter-2 M, letter-3 not A, cue null, image NIJM.
- (f) letter-1 N, letter-2 not M, cue null, image NAM.
- (g) letter-1 not {F, R, P, N}, cue null, image NUM.

The net is represented as a tree, with roots at the top, and with paths from root to leaf representing Ps. For ease in mechanical formatting, rather than having a parent node centered above and between its descendants, it is directly above the leftmost one. One should imagine that each node is connected by a line to all nodes on the next level below it that don't have anything above them. A negative test (test for absence) is represented by "else". The "else" is meant to stand for the negative of all the tests that have preceded it on the same level, except that its scope is bounded by any other "else", if such occurs. In the RHSs, the "fired" signal is implicit, the cue is marked by "=>", and the image, by "I ". "-" is used in cues to indicate a "don't-know" position. "?" indicates the "don't know" (DUNNO) image. The LHS tests use noticing order, that is 1, 2, and 3 (as marked by the tree-level labels on the extreme left) stand for tests on letters 1, 3, and 2. A space in a test implicitly means that the test is the same as the test in the first non-empty column to the left of the space. Notice that the order of the first three columns corresponds to the order of the three stimulus syllables.

To illustrate how a net is built up, we go through a sequence in which the following three pairs are partially learned: FIB/NUM, RAM/POR, and PEK/NAM (the same ones as used in the example net above). The net diagram following a pair is the state of the net after the pair has been processed.

FIB/NUM			
1:	F		else
	=>N--		=>---
	I ?		I NUM

The net is initialized (by a special P that recognizes the lack of net behavior) to minimally recognize the pair. Note that the second-column P serves two functions, balancing the F test (i.e., catching stimuli that miss the F test), and minimally recognizing "N" and emitting NUM as an image. All tests are balanced by such "else" Ps as the net grows.

```

RAM/POR
1:  F      else
    =>N--  =>P--
    I ?   I NUM

```

On the first try for this pair, the absence of cue is noticed in response to the R of RAM, so that the cue P-- is added to the second P and the pair is fed into the net again.

```

Internal RAM/POR
1:  F      R      P      else
    =>N--  =>P--  =>---  =>---
    I ?   I ?   I POR  I NUM

```

This was actually accomplished in two steps: first, it was noticed that POR and NUM are incompatible (the same incomplete cue can't apply to both), so that more discrimination of the stimulus was necessary, resulting in adding the "R" P; second, a P was added to discriminate the P-- cue. The detection and use of response-image (POR and NUM) incompatibility differs from the original EPAM. The original stored enough of the stimulus as an "image" at a node to detect immediately, on presentation of a new stimulus, that more discrimination on the stimulus was necessary. That is, it stored the exact equivalent of the piece of the correct stimulus that should have been tested in reaching the node. The present implementation has accomplished the same action without that stimulus image storage, and assumes instead that the two responses are available in full for comparison, along with the degree of completion of the cue. It thus uses less long-term memory for exact syllables, and uses instead the comparison of a "recently-heard" syllable with a "recently-spoken" one. This is the major significant difference between this implementation and the original.

A second feature of the action represented above is somewhat obscured by that representation, namely, the retention of the ! NUM image in the "else" P. As a P is added, a previous P is actually split, and some information from the old P is retained in parts of the two new Ps. The old image is always retained in the "else" P, but the old cue information may go either way, according to whether the new test being added (in the present case, "P") is the same as the cue or not. (Recall that a single P may store both cue and image - in the case at hand, the cue is irrelevant to the pair being focussed on.) The way the old image goes during the split may result in errors later in case the newly-added test is still not enough to distinguish the two responses. Having it go to the wrong place in a few rare cases was judged better than discarding it completely, in terms of speed of learning (this may be a difference between EPAM and the original EPAM). In any case the presence of the wrong reply will not interfere with future learning, but will just be an extra copy of that image in the net. This will be illustrated in connection with the third pair.

C.

An Example of the Building of Net Productions

EPAM

```

PEK/NAH
1:  F      R      P      N      else
    =>N--  =>P--  =>N--  =>---  =>---
    I ?   I ?   I POR  I NAH  I NUM

```

Here again, two steps are involved in the processing. First, the "P"-test P had no cue previously, so the cue was added to be appropriate to the response. Then the stimulus was presented (internally) and the wrong reply (NUM) led to the splitting of the "else" P, with the result shown. The image on the "else" P is now wrong, as anticipated above, and the pair FIB/NUM has thus been "forgotten". It would not have been forgotten if it had been, say, FIB/JUM.

The end of the list of pairs to be learned has been reached, so we start over with the first pair.

```

FIB/NUM (program generates reply NAH)
1:  F      R      P      N      else
    =>N-M  =>P--  =>N--  =>---  =>---
    I ?   I ?   I POR  I NUM  I NUM

2:                                     M      else
                                     =>---  =>---
                                     I NUM  I NAH

```

Note that now the cue for the F P has been extended, and that the tests for the N Ps include tests on the second letters. When it was discovered that the reply was wrong, it was also noted that the response and the reply were compatible as far as the cue could distinguish, so that the only action necessary was on the reply-producing part of the net (as opposed to the stimulus-recognizing part), namely, on the producer of the cue and the producer of the reply. The extension of the cue forced the extension of the tests to include the second letter, M, of the response (recall that the noticing order is 1-3-2). Note further that we now have two incorrect reply images, the original NUM, and the incorrect NAM. NAM is incorrect only because it has two letters in common with NUM. Another pass will be required to extend the "P"-test P cue and make the further discrimination in the image producer. Since that is fairly straightforward, we will omit it here, but include it in the detailed trace in Appendix C, which the reader may be able to understand after studying Section D. The end result is the net given at the beginning of this section.

D. Details of EPAM

This section gives details on the operation of EPAM. First we present an abstract version of the PS and discuss some features in more detail. Then we give the meanings of the predicates that constitute the Ps. When specific Ps are referred to, the reader may find them in Appendix A, the program listing. A cross-reference of predicates is given in Appendix B. The reader may also study the detailed trace of EPAM on the three-pair test (Section C), in Appendix C. There is a summary of that trace showing the type of P firing (according to the first letter of the P's name), at the end of Appendix C. After each pass over the three pairs, the net Ps are displayed; the final net Ps are included in the cross-reference in Appendix B.

EPAM has been tried on four different lists of pairs. Summaries of its behavior appear near the end of Appendix C, for the three-pair test, and in Appendix D. The latter appendix contains three tests: a seven-pair list to be discussed in Section F, a nine-pair list, and a six-pair list that actually represents a list of six syllables - each one appears as both a stimulus and a response.

D.1. How the program works

An abstract representation of the EPAM PS is given in the form of abstract Ps (APs) in Figure D.1. The following paragraphs describe EPAM in a general way, referring to specific APs in that figure. Syntax conventions for APs are given in Chapter IV, where the syntax is somewhat more elaborate than the simple APs here. For the present, the following description of APs should suffice. An AP is abstract in the sense that it represents several actual Ps and uses descriptive elements rather than exact predicates and variables. Underlining is used where there is a particularly large step in terms of actual Ps. Non-underlined elements correspond almost exactly to actual conjuncts in the actual Ps. There are about half as many APs in the figure as there are Ps in EPAM (16 as opposed to 41).

The normal operation of EPAM is a simple cycle that processes a stimulus-response pair. It starts by taking in a stimulus, allows the net to recognize that stimulus (which always occurs except when there is no net, at the very beginning), erases superfluous items from that net firing, and allows the net to fire on any cue information that the previous net P has asserted. If anything fires using the cue, an image is produced, which becomes the program's reply. The reply is matched to the response typed by the user, and if it is correct, EPAM is ready for the next cycle.

The basic part of that stimulus-response cycle is represented by APs Fb-Fc. AP Fa is the initialization done when there is no net. The matching of response to reply (APs Ra-Rd) can have four results, of which three are errors (Rb-Rd) requiring further action. The action is in three conceptual pieces: diagnosis, patching, and sometimes repeating the stimulus-response pair internally to ensure that everything is now all right.

Diagnosing an error can have three outcomes. If EPAM's reply was DUNNO ("don't-

- % Executive; 3 Ps, F1-F3; erasures mentioned are 5 Ps, E1-E5 %
 Fa: no-net -> repeat-stim-input & add-prod(type Pa) & add-prod(type Pb);
 Pa: specific-letter-1-of-stim -> cue(letter-1-of-resp) & null-image;
 Pb: not specific-letter-1-of-stim -> null-cue & null-image;
 Fb: cue-prod-fired -> erase-stim;
 Fc: image-prod-fired -> erase-cue & cue-from-image-prod & reply;
- % Test reply; 4 Ps, R1-R4 %
 Ra: response & reply-match -> reply(ok);
 Rb: response & reply-dunno -> simple-patch-of-image-of-reply-prod;
 Rc: response & reply-wrong -> test-compat-of-reply-and-response-relative-to-cue;
 Rd: response & not reply -> add-to-cue & re-stim;
- % Diagnose difficulties; 11 Ps, R5-R10A %
 Re: response-and-reply-and-cue-compatible
 -> extend(reply prod) & possibly-add-to-cue(cue prod);
 Rf: not response-and-reply-and-cue-compatible
 -> extend(cue prod) & extend-stim & re-stim;
 Rg: re-stim -> repeat-stim & repeat-resp;
- % Change cue; 3 Ps, C1-C3 %
 Ca: add-to-cue -> patch-up-cue-in-rhs;
- % Examine LHS to prepare for split; 5 Ps, C4-C8 %
 Cb: extend & lhs-term(neg or pos) & number-of-letters-tested-in-lhs -> split-prep;
- % Preparation for split; 8 Ps, S3-S7 %
 Sa: split-prep & not extend-stim & match-corresponding-letters(stim,resp)
 -> split-prod(two ways of ordering pos and neg cues);
 Sb: split-prep & extend-stim & match(response-letter,beginning-of-cue)
 -> split-prod(two ways, may use old cue or not for neg part);
- % Split; 2 Ps, S1-S2 %
 Sc: split-prod(neg)
 -> split-with-pos-part-getting-new-image-and-neg-part-getting-old;
 Sd: split-prod(pos)
 -> add-to-cue & split-with-different-lhss-but-otherwise-like-preceding-P;

Figure D.1 Abstract productions for EPAM

know"), there is simply a modification to the faulty RHS of the image-producing P (AP Rb). If there was no reply at all, only one thing can be immediately at fault: the cue produced on recognizing the stimulus was inadequate, not long enough (AP Rd). This is fixed by making the cue longer, as described below. An example of how this happens is after the second presentation of FIB/NUM in the three-pair test discussed in Section C (see the last net displayed in that section). The P that recognizes FIB has an adequate cue, N-M, but the other P that aims at a response syllable starting with N (the POR-test P) has cue N--, which is not long enough to fire any P in the net.

The third outcome of an error diagnosis (AP Rc) is to initiate a match of the cue, the reply (image), and the correct response, to see if the cue is at fault or whether more discrimination has to be made on the cue by the image producer. Two results of this match indicate two conditions: stimulus generalization and response generalization. Stimulus generalization is detected by incompatibility of the cue, reply, and response (AP Rf). Two stimuli have been "generalized", seen by the net as the same. For instance, if NUM were the reply and POR the response, no cue could be right for both, indicating that there was not enough discrimination of the stimulus. The length of the cue is taken into account, for instance, to determine that N-M is compatible with both NAM and NUM. Response generalization means that a cue is no longer sufficiently detailed to distinguish two similar responses, which are now seen as one (AP Re). For instance, in the three-pair test in Section C, after the first pass over the pairs, there are two N-- cues, both "pointing" at a P whose image is NAM; one wants to result in NUM, though. The patch is to lengthen the cue and eventually force lengthening of the LHSs of the image-producing Ps. In the case of NAM and NUM, lengthening of the cue N-- has to be done twice, which requires another pass over the set of pairs, to be realized.

There are two major kinds of patching that can be done, as sketched in the preceding description of error diagnosis. First, a cue may be extended. This is done (AP Ca) by simply making the appropriate RHS emit a cue that is longer by one letter than the previous one. This lengthening uses letters from the correct response. Second, a cue P or an image P is split by lengthening its LHS in two different ways. One half of the split has a positive test on a letter, and the other is the balancing, "else" type of test (APs Cb, Sa-Sd). This splitting will be described in more detail below.

After the diagnosis and patching have been completed, in cases where a cue is extended or an LHS is changed, the stimulus and response are repeated internally. The patches are really intended only to partially fix up any problem, and the best way to complete the job is to retry the pair, as opposed to examining the net Ps more closely to statically determine if everything is all right. The patches first remedy things having to do with the stimulus half of the pair, then treat the image-producing half through the repetition of the pair. The repetition (re-stimulation) is achieved in a way corresponding to AP Rg, but several Ps are used, to keep the sequencing under control. One P fires to reproduce the original stimulus, and another fires later to reproduce the response. Signals for each are represented in the interim by predicates whose names start with "OLD", by convention. This internal re-stimulation was not used in the original EPAM as far as I can tell.

We now discuss in detail the information that is kept in Working Memory on each net P, and how it is used. All information is in four predicates: LHSPREL, LHSTERM, RHSIMAGE, and RHSCUE. These split the LHS and RHS each into two segments. The two for LHSs divide them in such a way that all real changes are made to LHSTERM, while LHSPREL is just the accumulation of tests from LHSTERM that are known not to require further changes. RHSIMAGE keeps the image part of the RHS, while RHSCUE keeps the cue.

The following examples clarify the mechanics of splitting LHSs. First, consider splitting

P1, LET1(V1) & NOT F(V1) => FIRED(P1) & ...

Suppose we want to make a further test on the first letter, to see if it is an M. Then the result would be:

P1: LET1(V1) & M(V1) -> FIRED(P1) & ...;
 P2: LET1(V1) & NOT F(V1) & NOT M(V1) -> FIRED(P2) & ...;

P1 is referred to as the positive half of the split, P2, as the negative half. For the original P1, LHSPREL is [(LET1 V1)], LHSTERM is [((NOT (F V1))) NEG 1]. After the split, we have for P1 that LHSPREL is the same, while LHSTERM has become [(M V1) POS 1]; at the same time, P2 also has the old LHSPREL, but LHSTERM is [((NOT (F V1)) (NOT (M V1))) NEG 1].

If we were to further split P1, adding a test for T in the third position, we would have:

P1: LET1(V1) & M(V1) & LET3(V3) & T(V3) -> FIRED(P1) & ...;
 P3: LET1(V1) & M(V1) & LET3(V3) & NOT T(V3) -> FIRED(P3) & ...;

At this point, LHSPREL for P1 and P3 is [((LET1 V1) (M V1) (LET3 V3))], and LHSTERM is [((T V3)) POS 2] and [((NOT (T V3))) NEG 2], respectively.

This example doesn't touch on the details of how RHS information migrates during such splits, so we briefly discuss that now. Recall that some care has been taken to correctly place old information rather than simply discarding it. The old information has little to do with the immediate error situation, but tests are necessary to determine how it might interact with new discriminations.

In all splits, the image from the old P always goes to the negative half of the split, and the current reply is the image for the positive half.

In splitting a P that recognizes a stimulus (AP Sb, Ps S6-S7), the cue is always a minimal cue for the known response, and the only question is whether to put the old cue from the P's RHS on the other half of the split, or to leave that cue empty. If the old cue is similar (same first letter) to the new cue then it is discarded and an empty cue is placed. On the other hand, if the old cue is different, then it is likely to be correct to put it on the negative half of the split - that P will perhaps respond to the old stimulus (because the LHS has been extended) and will then emit a cue to the matching reply.

In splitting a P that discriminates a cue to produce an image for a reply (AP Sa, Ps S3-S5A), it is necessary to compare the stimulus and response, to determine whether the same P (the positive half of the split) will serve for producing both cue and reply. For this, it is only necessary to test the letter of the two that corresponds to the letter position at which the LHS is being split. If the letters match, the old cue goes to the positive half of the split, but if not (as is usually the case), the old cue goes to the negative half. The other half of the split (negative or positive, respectively) always gets an empty cue.

D.2. Meanings of EPAM predicates

Before presenting the full details of the predicates, we summarize the important representational conventions. The predicates LHSPREL, LHSTERM, RHSCUE, and RHSIMAGE are used to store information about net Ps. ADDPROD, REPPROD, and REPRHS are used to modify the Ps.

Stimuli and responses are typed in by the user in the form (STIM xxx) and (RESP yyy). These are converted into suitable Working Memory instances by PSMacros, Lisp functions that are used to expedite operations that would be very awkward otherwise (there is further discussion in Section E.1). STIM is converted into LETn's (n = 1,2,3) and specific letters, plus STIMCHK and STIMWORD instances. For instance, the list of instances for a syllable with third letter J might include (LET3 L3-1) and (J L3-1). Here "J" has become a predicate meaning "equal to J", and "L3-1" is some arbitrary token that stands for an object with predicates LET3 and J true of it. The representation splits apart the position and letter values of a token because they are sometimes not both present in Working Memory and sometimes tested separately. If Psnlst had somewhat more expressive power in its match (using constants is somewhat awkward), a more concise representation would be used. RESP is converted more simply, to an instance of RESPONSE, described below. A third PSMacro, CUE, expands a cue into letter representations like those of STIM, plus an instance of CUETRIPLE, described below. More complete examples of PSMacro expansions are given at the beginning of the program listing in Appendix A.

ADDPROD(pred,prec,comnt,lhs,rhs) add a P (prod) with comment comnt, LHS lhs, RHS rhs, and preceding P prec (if prec is not a P, as is the case in F1, prod is taken to be the first P of module prec); a Penlst primitive that asserts predicate ADDPRODP(pred).

ADDTOCUE(p,l1,l2,l3) add to the cue of P p, according to the letters l1,l2, and l3.®

COMPATNEG(x) result of compatibility test was negative.

COMPATPOS(x) result of compatibility test was positive.

COMPATPOSA(p,w,l1,l2,l3,p2) data to be used after the COMPATTEST is answered positively (part of this information is used for negative answers also); p1 is the reply P, p2 is the cue P, w1 is the image that might be used (from the response), and l1, l2, and l3 are the letters of the response.

COMPATTEST(l1,l2,c1) test a letter of the response, l1, against a letter of the reply, l2, except compare doesn't matter if c1 is 'XX'.

CUEPROD(p) p is the P that fired giving the cue.

CUETRIPLE(p,l1,l2,l3) P p has cue triple l1, l2, and l3; the l's are 'XX' if the cue is incomplete in that position.

EXTENDLSRS(p,im,l1,l2,l3) extend the LHS and RHS of P p, according to known properties of the P, using letters l1, l2, and l3.

EXTENDSTIM(p) signal that P p is being extended as a stimulus P, requiring slightly different treatment from extension otherwise.

FIREDP(p) P p has fired.

IMAGE(i) i is the reply image stored with a P that fired.

LASTNEW(p) p is the last new P added.

LETn(x) n is 1, 2, or 3; the 1st, 2nd, or 3rd letter of a stimulus or cue is represented by the token x.

LHSPREL(p,prel) P p has "prelude" prel; for network Ps, the prelude is the part of the LHS that is presently not subject to change.

LHSTERM(p,term,sign,len) P p has "terminal" term; for network Ps, the terminal is the final test of the LHS;

® All letter orders in predicates are the natural 1-2-3 order, not the 1-3-2 noticing order.

- sign indicates whether it is positive(POS) or negative(NEG); len indicates the length of the LHS (1, 2, or 3).
- OLDCUETRIPLE(I1,I2,I3)** I1, I2, and I3 are parts of an old CUETRIPLE; this is done to avoid confusion between two cuetriples asserted by two net-P firings.
- OLDEXSTIM(x)** indicates that the stimulus P has already been extended for this stimulus-response pair.
- OLDRESP(r,I1,I2,I3)** this is used to save RESPONSE while re-cycling through the net; it mainly prevents spurious firings of R4.
- POSSADDTOCUE(p,I1,I2,I3)** gives information which may be used to add to a cue, in the case of extending a LHS with a positive terminal test; i.e., that extension requires extending the cue so the extension will be used; it is not necessary for negative terminals of LHSs.
- REPLY(x)** gives the reply of the system (displayed externally).
- REPLYPROD(p)** P p produced the image that became the reply.
- REPPROD(p,com,lha,rha)** replace a P; similar to ADDPROD; asserts REPPRODP(p).
- REPRHS(prod,rha)** replace the RHS of P prod by rha; asserts REPRHSP(prod).
- RESPONSE(r,I1,I2,I3)** the response word, r, plus the three letters in it (comes in from external interaction).
- RESTIM(x)** signal that stimulus-response pair is to be fed through the net again, for further diagnosis.
- RESTIMHOLD(x)** holds the RESTIM signal until appropriate, mainly to prevent R5 from firing prematurely as a result of the STIMREM inserted in RHS of R4 (i.e., there is a conflict between R5 and E1, E2, etc. which is here resolved by using the RHS order assumption).
- RHSCUE(p,c)** c is the cue of the RHS of P p.
- RHSIMAGE(p,i)** i is the reply image for P p.
- SPLITPREP(prod,testpre,testterm,imj,size)** perform tests in preparation to split P prod; the three middle arguments are explained at SPLITPROD; the size gives the number of letters being tested in P prod's LHS.
- SPLITPROD(prod,testpre,testterm,imj,cuepos,cueneg)** P prod is to be split; if testterm is NEG, it is to be split negatively, which means, the test of a letter given in testpre is to be added to a negative test for one part of the split, and as a positive test in the other; in either case, no change is made to the LHSPREL of the P; if testterm is not NEG, it is to modify LHSTERM, while testpre modifies LHSPREL; imj is the image to be put in the positive part of the split; the last two arguments give pieces of cues to be attached to the respective parts of the split.
- STIMCHK(I1,I2,I3)** gives the tokens for the letters of the stimulus, for use in erasing them properly after their use in the net.
- STIMREM(I1,I2,I3,type)** stimulus tokens are to be erased; type gives an indication of where the request originated, since stimulus tokens are re-inserted for RESTIM.
- STIMREMREM(x)** causes all old STIMREM's to be cleaned up, because in case of RESTIM, don't want things nuked as soon as they're inserted from net Pa firing for their second time.
- STIMWORD(I1,I2,I3)** gives the three letters of the stimulus word.

E. Some Issues Raised by EPAM

E.1. Production systems issues

PS control:

For the most part, control in EPAM is simply by evocation according to Psnlst's event order. Several of the P groups (especially those represented by a single AP, Figure D.1) represent a selection, where a single P from the group fires to perform the selection appropriate to the case at hand. The compatibility-test Ps (R6-R9E, the RHS of AP Rc) are a set from which several may fire in a situation, with results independent of firing order, to perform a test on all three letter positions of several syllables. The results of the arbitrarily-ordered firings are polled by R6 and R9, amounting to answers "all tests positive" or "at least one test negative". The erasure Ps (E1-E5) similarly perform in scattered order. In a couple of places, data is renamed by storing it under a different predicate (i.e., the "OLD" predicates and RESTIMHOLD), so that it doesn't interfere, but can be re-asserted at some later appropriate point (i.e., by Ps F3, R4, R10, R10A, or R5A).

Use of PSMacros:

STIM and RESP are PSMacros, that is, they are expanded to a list of predicates to be used by Psnlst (see the beginning of Section D.2). This procedure is justified by thinking of them as perceptual processes, dealing with input on the character level, and thus more easily handled and more appropriately modelled in Lisp than in Psnlst. In any case they could be done away with at the cost of requiring either more Ps (which would still have to invoke primitive Lisp functions) or more typing for the user. The use of a macro (CUE) for constructing parts of RHSs of net Ps is not so easily justified, but this started out as a programming convenience, and remains as such, even though it could be coded as Ps without the use of primitive Lisp functions. This extra requirement on the PS would add more complexity to F1, S1, and S2, for instance necessitating adding several new predicates for the bookkeeping aspects. It has seemed reasonable to suppress that level of detail, retaining more emphasis on the central issues.

The operators used in building Ps:

There are several operators that Psnlst doesn't have, that would be useful for the kind of building of Ps that EPAM does:

- a. update an RHS conjunct;
- b. extend an RHS with more conjuncts;
- c. extend an LHS with more conjuncts;
- d. split an LHS, extending it in two different ways to become two different Ps.

The information that is kept on Ps to allow them to be updated is: information on specific conjuncts of the RHS; information on a subset of conjuncts of the RHS; and "head" versus "tail" in the LHS, where the "tail" may be replaced in a splitting operation. EPAM

also keeps information on the names of variables in the LHS, but this could be dispensed with by naming variables according to positions in stimulus or response words, which would ensure uniqueness.

Using the above new operators on RHS conjuncts, C1-C3, R2, S1, and S2 would become simpler. Using the LHS-splitting and LHS-extending operators, S1 and S2 would become somewhat simpler but the processing done by them would have to be rearranged so that the split is done at the start of the process, with succeeding steps simply extending the LHSs of the results of the split; this is in place of the current process which collects the contents of the split and then does it. Note that P F1 still requires the adding of full Ps as done with current Psnist operators.

Working Memory information on net Ps:

As we have discussed above (Section B and at the end of Section D.1), EPAM keeps and uses a variety of Working Memory information on Ps. One issue with respect to this is that, following human PS models, we would want to store all such longer-term information as Ps rather than in the Working Memory. Given the simple nature of the information used (at no time is a P broken down and examined in an arbitrary way), and given that it is used only as transient information (only when the relevant P has recently fired, where "recent" is within two cycles of the net Ps), objections to EPAM on these grounds can be easily corrected by storing with each P, in its RHS, the information on it. (There might also be defined a set of primitives that allow a program to fetch parts of Ps on demand, rather than having to use special storage.)

A second issue is that no matter where the information is stored and when it is used, it might be inappropriate to have such information, and furthermore it might be wrong to hold that existing Ps can be modified at all. The psychological basis for this is the assertion that Ps are never deleted from human long-term memory (we lose access because of ill-formed or too-specific LHSs) and thus are difficult to modify as in our EPAM model. Further, since we have no idea how Ps are represented internally in humans, perhaps information about what is encoded could not be available in as usable a form as in our model. A compromise might be that existing Ps could be modified by extending LHSs and RHSs but not by such drastic actions as replacing existing condition or action elements. At the moment, it seems that operating under these constraints makes EPAM much more difficult to model, but it remains a question for further research.

E.2. EPAM issues

This implementation aims at minimizing the number of Ps and the number of tests made on a letter position. This results in using the "else" Ps instead of having every test made as a comparison to a definite letter. Changing this convention might remove some of the stimulus and response generalization behavior, and interacts with the following.

A lack of a reply to a stimulus indicates that a cue needs to be extended. This results from the way a partial cue is represented. Only as many "LET" predicates are included as there are definite letters emitted. If this were changed, i.e., emitting LET without a definite corresponding letter predicate, some other means would have to be

developed to test for the need for an extension of a cue, because "else" Ps would always catch them. That would include some comparison between the cue triple and the contents of the LHS of the P that fired (as it is, no such test is ever done; only tails of LHSs are examined in detail, and only when being extended). Also, the change would introduce the possibility of conflicts between the net Ps, making more than one true at one time. Note that this alternative suggestion involves using specific knowledge of the length of a syllable, to allow dummy tests to be included. This may or may not be justified, especially given that a more general task involves syllables of varying lengths. The most important point here is the trade-off between storing partial cues and examining LHSs in more detail. Storing partial cues results in behavioral cues that circumvent the need to do more complex tests.

When Ps are being split, some degree of guessing goes on, in that information from the P being split is carried over to one or the other of the new ones, with no guarantee that the result is correct. The result is wrong only in the case that further discrimination is needed anyway, a fairly rare case, and a wrong guess does not have serious consequences for the net at some later time. Not retaining as much as possible in this situation gives the appearance of stupidity to the program's responses, more so than seems reasonable. But as discussed at the end of Section D.1, carrying over these guesses involves specific testing that may appear implausible, or at least not plausibly learned under ordinary conditions.

F. A Comparison to Waterman's EPAM2

This section compares EPAM to another PS implementation of EPAM, the EPAM2 PS of Waterman (1974). This provides an excellent opportunity to contrast the different programs that result when somewhat different architectural assumptions are followed in a PS design. EPAM2 is written in the PAS-II PS, which uses a linearly-ordered PS and a Working Memory also linearly ordered, a sharp contrast to Psnlst's exclusive use of event (memory recency) order for Working Memory and unordered PS. These assumptions have an affect on complexity of LHSs, on complexity of added Ps, and on the number of Ps in the entire system. But as we shall see in the following, major differences are due to specific EPAM design considerations that are only indirectly related to PS architectural features.

EPAM2 represents learned associations in a way similar to EPAM, namely as a list of Ps containing primarily tests on letters and actions such as emitting memory cues and reply images. The order of the list of Ps is significant, however, in determining that some Ps are tested before others, and thereby can prevent the others' firing. More recently added Ps are placed above older ones, enabling a new P to correct errors made by older ones and to make necessary additional discriminations by having more specific tests (but not necessarily logically exclusive tests) than the older ones. The net has Ps of two distinct types, one for discriminating stimuli to produce cues and the other for discriminating cues to produce replies. The two nets are not kept distinct either by RHS form or by location within the list of net Ps, so that a confusion phenomenon can result where a cue is taken as a stimulus, and vice versa.

Ps are extended in a way very dependent on the order of elements within Working Memory. There is a special P action that marks which elements were used to match the LHS of the P when it fires (called variously MARK, USED, and OLD). This allows a P to be constructed as an extension of a P that just fired by using the same elements as it did, plus a new element picked from things in Working Memory that were not used. These unused elements (in particular, letters or syllables) are in a particular order (the EPAM noticing order) so that the one at the front of the Working Memory list is quite appropriately selected and used.

The action that adds a P to the list of Ps, PROD, has a couple of distinctive traits. It works by picking up from Working Memory all elements with the tags COND and ACTION, forming them into a P, and placing it in the P list. The placement is according to a pattern argument to PROD, such that a new P can be placed just before another P having that pattern. For instance, if a new P is intended to mask out (take precedence over) a P that performed a particular undesired action (a wrong reply), it can be done by using that wrong reply as the PROD pattern.

EPAM2 builds a net in which all tests are specific and positive, contrasting with EPAM's use of negative tests, which can turn out to match a set of letters. Rather than adding, in advance, extra Ps to balance newly-added ones, as EPAM does with its P-splitting operation, EPAM2 has a default P that sits at the end of the list of net Ps, with a condition general enough to match all cases where none of the net Ps succeeds. Also, EPAM2 stores a complete copy of a stimulus in the RHS of a stimulus-testing P, which can

be used to determine if the P has actually matched the stimulus that it was intended to. This is somewhat more than the original EPAM stored, violating the EPAM principle of storage of minimal and partial information, and it avoids the compatibility-testing approach necessary in the present EPAM to determine whether to modify the stimulus P or the cue P in case of error. (The test in EPAM is a special case of the more general problem of matching two objects that appear similar on the basis of the minimal cues stored in the discrimination network. In the general case, it is even less desirable to store in the RHS of a P a complete image of the stimulus. This will be illustrated in Chapter IV of this thesis.)

The result of these design features is a PS that is more concise than EPAM, both in number of Ps and in length of listing: half the number of Ps (20 vs. 41) and about a seventh the number of lines (but EPAM2 has no comments). As we shall see below, EPAM2 doesn't have some of the functional units that EPAM does, and EPAM's tests tend to be more intricate, making individual Ps more complex, because the net being built is more intricately designed. A listing of EPAM2 is given in Appendix E, but it is not expected that the reader will be able to follow its details without reference to Waterman's description (1974).

EPAM2	STIM	REPLY 1	REPLY 2	REPLY 3	RESP	EPAM	REPLY 1	REPLY 2	REPLY 3
PAX	?	CON	CON	CON		--	? (NG)	CON	
BEK	?	MAB (SG)	LUQ	LUQ		--	-- (IC)	LUQ	
CIT	CON (SR)	DER	DER	DER		--	DER	DER	
BUK	LUQ (SG)	MAB	MAB	MAB		LUQ (SG)	LUQ (SG)	MAB	
NAL	?	LUQ (RG)	LEQ	LEQ		--	PEO (SG)	LEQ	
REB	?	MAB (RG)	MOL	MOL		LEQ (SG)	MOL	MOL	
NOJ	LUQ	PAX (SR)	PEO	PEO		LEQ (SG)	PEO	PEO	
		(SG RG)							

Keys: SG = stimulus generalization error; RG = response generalization error; SR = stimulus-response confusion; IC = insufficient cue; NG = net growth lost information; ? = program says "?" or "don't know"; -- = program makes no reply.

Figure F.1 A comparison of behaviors on a seven-pair test

Figure F.1 compares the behavior of the two PSs on a set of seven pairs taken from Waterman's paper. There are three essential differences between them. First, EPAM seems slightly slower in learning some of the pairs. This is due to its less specific "else" tests - these negative catch-all tests are slower to converge to the right specific-letter tests, and when they do change, cue information is discarded. Second, EPAM has no response generalization behavior on this set of pairs (although it did on the example in Section C). In the place where it had insufficient cue (IC in the table), it would have exhibited response generalization - the net had grown beyond the initial cue for the pair. Third, EPAM has no stimulus-response confusion because the PS distinguishes the kind of action it takes according to where in the pair-presentation cycle it is - it knows when it's appropriate to use a memory cue and when to use a reply image. This is made necessary by the use of a single net P format, since the same kind of P fires at both points in the cycle.

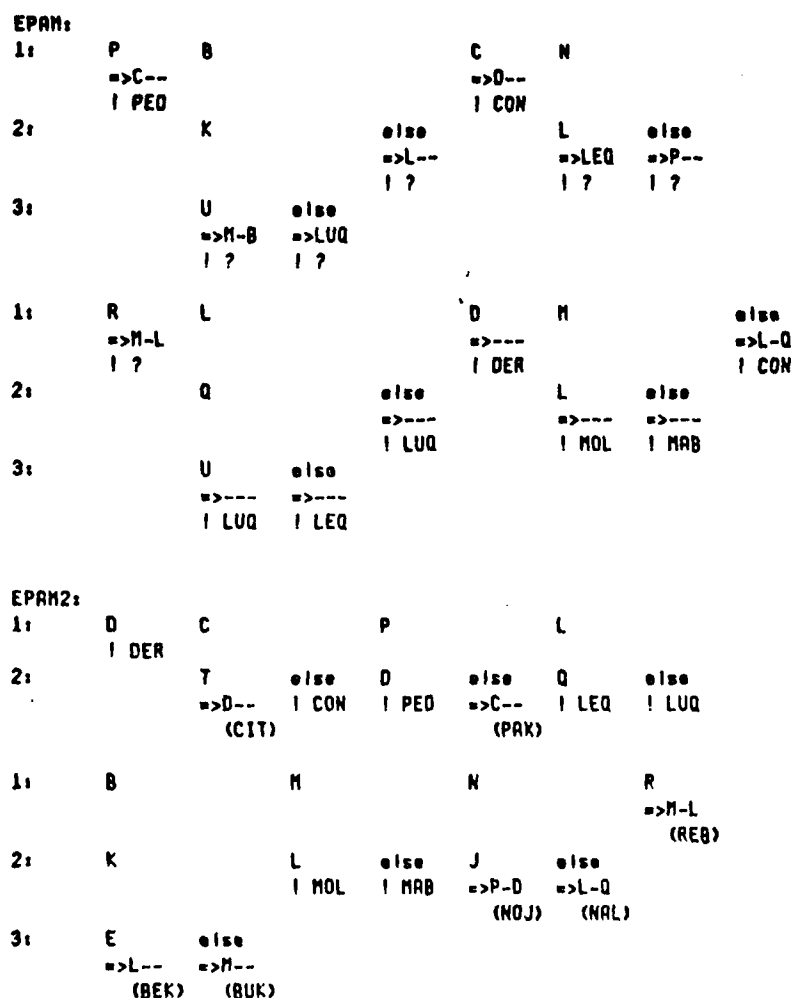


Figure F.2 Networks produced by the two programs

Figure F.2 gives the networks produced by the two programs for the seven-pair test in Figure F.1. The notation for the EPAM2 net is slightly different. Where an "else" occurs, it means a lack of a test on that syllable position rather than an explicit negative test as for EPAM. Note that this "else" test is achieved simply by placing the more specific P before the "else" one in the ordered P list. I have added syllables in ()'s to indicate the extra stimulus image information stored in EPAM2's net. The order of the columns in EPAM's net corresponds to the order of the stimulus syllables, then the response syllables (the latter are not quite in a corresponding sequence because of double use of some Ps as stimulus Ps). The order of the columns in EPAM2's net, however, are given in an order close to that in which order that they appear in the constructed PS - this order is

significant to the interpretation of the PS, and it also indicates the order in which the Ps were added, with the more recent additions to the left. In listing the net, I have deleted eight redundant Ps from EPAM2's net - they are redundant because the PS interpreter would never look at them, since identical-LHS Ps exist in the order before them (the identical Ps have different RHSs though). There are three EPAM Ps that are useless, but they are included (they are all "else" Ps). Thus in terms of number of useful Ps, EPAM2 builds 14, EPAM builds 12; EPAM2 builds 21 Ps in all, EPAM, 15.

In one place, the EPAM2 net is shallower than the corresponding portion of the EPAM net, because the discrimination in question is done in EPAM2 (accidentally, it seems) by length of cue as opposed to contents. That is, because of the history of the pair learning, a one-letter cue, L--, evokes LUQ and a two-letter cue, L-Q, evokes LEQ; in EPAM, a full three-letter cue is required to distinguish the two (and in the original EPAM, to the best of my information). In another place, the EPAM2 net is deeper than the EPAM one (the tests on C in the 1 position), because EPAM2 requires two Ps to test for cue versus stimulus, where EPAM uses a single P.

A major behavior deficiency results from the EPAM2 net representation: it cannot learn lists such as PAX/CON, CON/LUQ, LUQ/PAX (the fact that this is circular is not a critical factor - the second PAX could be END with similar effect). By my hand simulation of the program, it would oscillate forever, trying to treat each syllable first as a cue P then as a reply P, always blocking out its previous usage by putting a new P before it. EPAM can learn such lists, as is evident in the summary included in Appendix D. There are apparently some advantages to the EPAM2 representation, though. Old Ps always preserve old behavior, in case conditions fall through more recent (and supposedly correct) ones - though this can never happen in EPAM2, by careful design. EPAM2 doesn't concern itself with the cue completeness discussed above in Section E.2. Also, the transfer of guesses is not necessary in EPAM2, while it is deemed so in EPAM (EPAM would learn more slowly without it). This is a result of EPAM2's carryover of information as a result of falling through the ordered list of newer Ps.

Figure F.3 tabulates the differences between EPAM and EPAM2 by grouping Ps by program function. EPAM2 Ps are simply numbered from 1 to 20. The number of Ps in each group is given in square brackets. The largest difference in that table is in the code to build and extend Ps: EPAM2 uses 9 Ps versus EPAM's 18 (combining the five lines in the table before the last one); 8 of EPAM's Ps are used in splitting existing net Ps, whereas the corresponding EPAM2 mechanism is having two distinct nets and using the fact that newly-added Ps mask old Ps, which mechanism uses no Ps in the EPAM2 code. That 8 Ps versus none is the largest single difference between the two PSs. Storing the full stimulus syllable as a TEST in net Ps saves EPAM2 6 Ps over the explicit comparison done in EPAM (1 P versus 7). EPAM's re-stimulation mechanism uses 4 Ps, corresponding to nothing in EPAM2. The other differences in numbers of Ps between the two (five other classes) amount to a total of 3 Ps difference, with the values of the differences between -2 and 2.

The largest difference in number of Ps between EPAM and EPAM2 is thus indirectly related to the use of an ordered-PS architecture. (Though it is largest, it is the combination of several groups of Ps in the figure, and it doesn't constitute a majority of the differences.) It is indirect because the component affected most is the network being built, not the basic EPAM PS itself - that is, there is a clear distinction in cost between

Functional units:	EPAM2	EPAM	
Input and distinguish STIM and RESP	1 2 3 6 9 10	[6] F2 F3 E1-E5	[7]
Initialize net	-	F1	[1]
Match REPLY and RESP	4 5	[2] R1-R4	[4]
SR Confusion	7 8	[2]	
Test compatibility of STIM and CUE	11	[1] R6-R9E	[7]
Set up to build reply P	12-14	[3] S3-S7 C4-C8	[8+5]
Set up to build cue P	16-18	[3] " "	
Build reply P	15	[1] S1-S2	[2]
Build cue P	20	[1] "	
Extend cue p	19	[1] C1-C3	[3]
RESTIM mechanism	-	R5 R5A R10 R10A	[4]
Total Ps		[20]	[41]

Figure F.3 A comparison by functional units

adding new Ps to an ordered list and gradually modifying existing Ps. Order is considered contrary to psychological evidence that recognition (matching a P condition) is a parallel action, not a search through a list of Ps. EPAM2's most serious defects as a model of EPAM are in the use of a dual network, which prevents it from learning lists; in the storage of the full stimulus image, avoiding a more complex comparison to assign credit for a mistake but violating EPAM's partial-storage principle; and in creating many redundant Ps instead of making incremental additions to the sensitivity of existing Ps. But EPAM2 has the more interesting behavior on the particular 7 pairs exhibited. EPAM's design position with respect to the negative-test "else" Ps is probably to blame for peculiarities of behavior. It decreases specificity of tests and thereby the likelihood of retaining information leading to response generalization. The use of splitting operations on existing Ps makes EPAM's design more intricate. (EPAM2's design is intricate in another way, though, since dependencies on PS order and on Working Memory order are implicit.) Since it decides definitely what to do with the various components of the Ps being split, it takes a more rigid position, and this is more likely to show up as peculiarities in behavior.

G. Conclusions

G.1. Summary of conclusions

The EPAM PS demonstrates the feasibility of using a PS to build a PS in response to the demands of a simple verbal learning task. Ps are added to a set of Ps representing a network, and Ps in that set are also modified by operations amounting to splitting and extending LHSs. Local use is made of knowledge of subcomponents of the Ps being modified. PSs are quite suitable for modelling EPAM discrimination networks. The particular EPAM design has focussed on minimizing the use of P storage and maximizing the use of structures already built. A comparison to a similar program makes it clear that there is a tradeoff between using knowledge about existing Ps and using an ordering on a list of Ps, although ordering is considered on independent grounds to be unsatisfactory.

G.2. Further research

There are several primary difficulties that require consideration beyond the present scope. In order to be plausible, an EPAM PS should be simple enough to be learned from some simpler basic knowledge. The present one is judged to be too complicated for that, but a careful analysis needs to be done to confirm it. Perhaps an attempt to do EPAM without modifying or using knowledge about existing network Ps should be carried out. This must go beyond Waterman's use of an ordering on the PS. Or perhaps the aim should be a more automatic and less awkward use and modification, along the lines presented here. That is, the splitting and extending might be made an automatic feature of the architecture, carried out when a new P fragment is presented to the PS. EPAM is in a sense rather fragile - if there is an error in an input pair, it may not be able to recover when given the correct version again, due to non-retractable and non-correctable modifications made to Ps. EPAM has significant promise as an effective way of minimally representing complicated objects or situations, but to become that, the basic process must be much more general.*

The following secondary difficulties pertain to the more narrow field of nonsense-syllable learning. In assuming three-letter syllables too strongly, EPAM became committed to very specific processing, whereas, it is now evident, a more general approach would actually simplify the representations and the PS. There is no provision in EPAM as it is for using a wider context to disambiguate multiple occurrences of a syllable in the same role within a list. Design issues with respect to specificity of tests, cue completeness, and preserving older information have been raised in the preceding sections. Finally, this effort has not aimed at reproducing behavior of humans as was the original EPAM. Adjustments for specific behavioral peculiarities, speed of learning (number of P firings), recency effects, and list-length effects could be considered, and seem to be readily attainable in PSs.

* Chapter IV presents some progress towards that goal.

G.3. Acknowledgement

Allen Newell suggested the alternatives in design with respect to keeping track of the knowledge already encoded as Ps.

G.4. References

Feigenbaum, E. A., 1963. "The simulation of verbal learning behavior", in Feigenbaum, E. A. and Feldman, J., Eds., *Computers and Thought*, pp. 297-309. New York, NY: McGraw-Hill.

Hayes-Roth, F. and Mostow, D. J., 1975. "An automatically compilable recognition network for structured patterns", Pittsburgh, Pa: Carnegie-Mellon University, Department of Computer Science.

Waterman, D. A., 1974. "Adaptive production systems", Complex Information Processing Working Paper 285. Pittsburgh, Pa: Carnegie-Mellon University, Department of Psychology.

EPAM

EPAM APPENDICES

Appendix A. Program Listing

```

BEGIN
    2 PS FOR (PAM 3)

EXPR (PAM3); BEGIN
    PSMACRO((PAM3); DOEND(PAM3);

    3 MACRO3 : (STM AND RESP ARE EXTERNAL ASSERTIONS)

    STIMF(I) -> EXISTS(I, I2,I3) & LET(I, I) & F(I, I)
        & LET(I2, I2) & I(I, I2)
        & LET(I3, I3) & I(I, I3)
        & STIMCH(I, I2,I3) & STIMWORD(V,I,I,I)

    RESP(XVZ) -> RESPONSE("XVZ,X,V,Z")

    CUE(PROD, I2,I3) -> EXISTS(C, I,C7,C3)
        & FOR EACH (N NON-XX, LET(CN) & (LCN)
            & CUE(1,PROD,X,I2,X,I3) WHERE X= IS Cn OR XX
        )
END

```

```

8 F : FIRED TESTS : CATCH WHICH NET PROO DID IT 8

F1: "INIT NET" = STIMCHN(L1,L2,L3) & NOT( EXISTS(P) & FIRED(P) )
    & STIMWORD(W1,W2,W3) & RESPONSE(RM1,M2,M3)
-> EXISTS(PW,M) & LET(I,L1) & STIMCHN(L1,L2,L3)
    & ADOPROD(PW,PAWM,"NET",<LET(I,V1),<W1,V1>),
    < FIRED,<QUOTE PW>,<IMAGE,<QUOTE,DUNNO>,>
    <CUE PW,M1,"XX,XX"> )
    & LHSREL(PM,<LET(I,V1)>)
    & LHSTERM(PM,<W1,V1>,POS.1)
    & RHSIMAGE(PM,DUNNO) & RHS(CUE(PM,<CUE PW,M1,"XX,XX">))
    & ADOPROD(PM,PW,"NET",<LET(I,V1),<NOT,<W1,V1>)>,
    < FIRED,<QUOTE PW>,<IMAGE,<QUOTE,DUNNO>,>
    <CUE PW,XX,"XX,XX"> )
    & LHSREL(PM,<LET(I,V1)>)
    & LHSTERM(PM,<NOT,<W1,V1>)>,NEG.1)
    & RHSIMAGE(PM,DUNNO) & RHS(CUE(PM,<CUE PW,XX,XX,"XX">))
    & LASTNEW(PM) & OLDDRESPRM1,M2,M3) & NEGATE(0)

F2: "STIM FIRE" = FIRED(P) & STIMCHN(L1,L2,L3)
    & IMAGE(W) & CUE TRIPLE(P,V1,V2,V3)
-> STIMCHN(L1,L2,L3,STIM) & OLDCUE TRIPLE(V1,V2,V3)
    & CUEPROD(P) & NEGATE(ALL);

F3: "CUE FIRE" = FIRED(P) & CUEPROD(Q) & OLDCUE TRIPLE(L1,L2,L3)
    & CUE TRIPLE(P,M1,M2,M3) & IMAGE(W)
-> STIMCHN(M1,M2,M3,DUNO) & STIMCHN(L1,L2,L3,CUE) & REPL(YW)
    & REPL(YPOW) & CUE TRIPLE(Q,L1,L2,L3) & NEGATE(ALL,-2)

```

3 PAGE 2 : R - RESPONSE [EXAMINATION RELATIVE TO REPLY 2

```

R1: "REPLY MATCH" = RESPONSE(W1 I1.2.3) & REPLY(W)
    >> REPLY(OK) & NEGATE(ALL);
R2: "REPLY MULL" = RESPONSE(W1 I1.7.3) & REPLY(W)
    & SATISFIES(SW MEQ DUNNO) & REPLYPROG(P) & INCUE(P2)
    >> REPLY(OK)
    & REPRIS(P, P.Y. REQ, "QUOTE P") & IMAGE, "QUOTE R" >> Q)
    & RMSIMAGE(P, R) & NOT RMSIMAGE(P, W) & NEGATE(I.2);
R3: "REPLY WRONG" = RESPONSE(W1 I1.7.3) & REPLY(R) & VNE(Q, W, R)
    & SATISFIES(SR MEQ DUNNO) & SATISFIES(SR MEQ OK)
    & CUEPROG(P) & REPLYPROG(Q) & CLRTIME(P.V1, V2, V3)
    >> COMPATTEST(I.1 CAR EXPLODE R.V1)
    & COMPATTEST(I.3 CAR EXPLODE R.V3)
    & COMPATTEST(I.2 CAR EXPLODE R.V2)
    & COMPATPROG(W1 I1.7.3.P) & NEGATE(ALL, 1-8);
R4: "REPLY ABS" = RESPONSE(W1 I1.7.3) & OLD CUE TRIPLE(V1, V2, V3)
    & CUEPROG(P) & NOTE(EXISTS(SR) & REPLY(P))
    >> STIMRENEW(V1, V2, V3, CUE) & ADDTOCUE(P, I1.7.3)
    & CLRTIME(P.V1, V2, V3) & RESTIMUL(D, OK)
    & NEGATE(ALL, 1);

R5: "RE STIM" = RESTIMUL(X) & STIMRENEW(I1.7.3.C)
    & SATISFIES(SC C Q "STIM") & CLRTIME(P.V1, V2, V3)
    & RESPONSE(R.M1 M2 M3)
    >> STIMRENEW(OK) & LET(I.1) & LET2(I.2) & LET3(I.3)
    & STIMCH(I1.7.3) & OLD RESIMUL(M1 M2 M3) & NEGATE(ALL);
R5A: "RE STIM-" = RESTIMUL(D, X) & RESTIMUL(X) & NEGATE(I.1)

```

```

(PB: "COMPAT ." = COMPATPOS(X) @ COMPATPOS2(Q.W/L1/L2/L3/P)

```

A

```

      & NOT( EXISTSV( V1, V2 ) & COMPATTEST( V1, V2, V3 ) )
      & NOT( EXISTSC( ) & COMPATG( C ) )
    >> EXTENDLDRSC( Q, W1, I1, L1, J1 ) & POSSADDTOCLUE( P, L1, L2, L3 )
      & REPLY( X ) & NEGATE( ALL );

R7: "COMPAT T-" <- COMPATTEST( W1, W2, C ) & SATISFIES( C, C EQ "X" )
  >> COMPATPOS( C ) & NEGATE( I1 );
R7A: "COMPAT T-A" <- COMPATTEST( W1, W2, C ) & OLDEXISTIM( I1 )
  >> COMPATPOS( C ) & NEGATE( I1 );
R7C: "COMPAT T OR" <- COMPATTEST( W1, W1, C ) & SATISFIES( C, C NEQ "X" )
  >> COMPATPOS( C ) & NEGATE( I1 );
R8: "COMPAT T-" <- COMPATTEST( W1, W2, C ) & NOT SATISFIES( C, C EQ "X" )
  & VHE( Q, W1, W2 ) & NOT( EXISTSI( I1 ) & OLDEXISTIM( I1 ) )
  >> COMPATG( C ) & NEGATE( I1 );
R9: "COMPAT -" <- COMPATG( C ) & NOT( EXISTSC( ) & COMPATPOS( C ) )
  & COMPATPOS( Q, W1, V1, V2, V3, P ) & STIMWORD( I1, L1, J1 )
  >> EXTENDLDRSC( Q, W1, I1, L1, J1 ) & EXTENDSTIM( P ) & RESTIM( Q )
      & OLDEXISTIM( I1 ) & POSSADDTOCLUE( "X", "X", "X", "X" )
      & NEGATE( I1, J1 );
R9E: "COMPAT -EM" <- COMPATG( C ) & COMPATPOS( C )
  >> COMPATG( C ) & NEGATE( I1, J1 );

R10: "EM RESP" <- REPLY( X ) & OLDRESP( R1, L1, L2, L3 )
  >> RESPONSE( R1, L1, L2, L3 ) & NEGATE( I1, J1 );
R10A: "EM RESP" <- OLDRESP( R1, L1, L2, L3 ) & NOT( EXISTSC( ) & REPLY( X ) )
  >> RESPONSE( R1, L1, L2, L3 ) & NEGATE( I1, J1 );

```

3 PAGE 3: C - CLE EXAM, TO DETERMINE WHAT TO FIX IN NET 3

```

C1: "CH CUE" = ADDTOCUE(P.1,1,2,1,3) & CUE TRIPLE(P.M1,M2,M3)
    & SATISFIES(M1,M1 MEQ 'XX') & BISCUE(P,Q,2) & RHS IMAGE(P,1)
    > REPHRASE('Y TIED,'QUOTE P','<IMAGE,'QUOTE Q,1'),
        CUE P.1,1,M2,M3'))
    & BISCUE(P,CUE P.1,M2,M3)) & NEGATE(1A);
C2: "CH CUE 3" = ADDTOCUE(P.1,1,2,1,3) & CUE TRIPLE(P.M1,M2,M3)
    & SATISFIES(M1,M1 MEQ 'XX') & SATISFIES(M3,M3 MEQ 'XX')
    & BISCUE(P,Q,2) & RHS IMAGE(P,1)
    > REPHRASE('Y TIED,'QUOTE P','<IMAGE,'QUOTE Q,1'),
        CUE P.1,1,M2,1,3'))
    & BISCUE(P,CUE P.1,1,M2,1,3)) & NEGATE(1B);
C3: "CH CUE 2" = ADDTOCUE(P.1,1,2,1,3) & CUE TRIPLE(P.M1,M2,M3)
    & SATISFIES(M1,M1 MEQ 'XX') & SATISFIES(M3,M3 MEQ 'XX')
    & SATISFIES(M2,M2 EQ 'XX') & BISCUE(P,Q,2) & RHS IMAGE(P,1)
    > REPHRASE('Y TIED,'QUOTE P','<IMAGE,'QUOTE Q,1'),
        CUE P.1,1,2,1,3'))
    & BISCUE(P,CUE P.1,1,2,1,3)) & NEGATE(1B);

```

```

C4: TEXT (S NEG) < EXTEND(SRSP,1,M1,M2,M3) < L1(STERNP,M,N,M)
    < SATISFIES(SUN EQ NEG,1) < NEGATE(1)
    < SPLIT(PREP,M,V1,NEG,1,1) < NEGATE(1)
C5: TEXT (S NEG) < EXTEND(SRSP,1,M1,M2,M3) < L1(STERNP,M,N,M)
    < SATISFIES(SUN EQ NEG,2) < SATISFIES(S EQ 2)
    < SPLIT(PREP,M,V3,NEG,1,2) < NEGATE(1)
C6: TEXT (S NEG) < EXTEND(SRSP,1,M1,M2,M3) < L1(STERNP,M,N,M)
    < SATISFIES(SUN EQ NEG,3) < SATISFIES(S EQ 3)
    < SPLIT(PREP,M,V2,NEG,1,3) < NEGATE(1)

```

```

C7: 'TEXT LS POS 3' = EXTEND(SRSP,(M1,M2,M3)) & LSTERN(MP,M,N,S)
      & SATISFIES(N,N,Q POS) & SATISFIES(S,S,Q EQ)
      & SPLITPRPPL('L12,V1,M3,V3',1,2) & NEGATE(1)
C8: 'TEXT LS POS 2' = EXTEND(SRSP,(M1,M2,M3)) & LSTERN(MP,M,N,S)
      & SATISFIES(N,N,Q POS) & SATISFIES(S,S,Q EQ)
      & SPLITPRPPL('L12,V2,M2,V2',1,3) & NEGATE(1)

```

REFRASE TRACES OF STIMULUS (OR CLUE) 2

```

(2) "STIM REM1" = STIMREM(M, 1, 2, 1, 3, 2) & LEFT(R, 1) => NEGATE(2)
(3) "STIM REM2" = STIMREM(M, 1, 2, 1, 3, 3) & LEFT(R, 2) => NEGATE(2)
(4) "STIM REM3" = STIMREM(M, 1, 2, 1, 3, 3) & LEFT(R, 3) => NEGATE(2)
(5) "STIM REM M" = STIMREM(M, M(X)) & STIMREM(M, 1, 2, 1, 3, 3)
    => NEGATE(ALL);
(6) "STIM REM M.W." = STIMREM(M, M(X)
    & NOT(KRISTOF, 1, 2, 1, 3, 2) & STIMREM(M, 1, 2, 1, 3, 3))
    => NEGATE(1);

```

3 PAGE 012 - SPLIT IT INTO TWO NEW ONES 2

```

S1: "SPLIT PROD NEG" = SPLITPROD(P,T,T,I,C,P,M)
  & SATISFIES(TT,TT EQ NEG) & LMSPREL(P,M,N,S)
  & LMSPREL(P,M) & LASTNEW(M) & RMSCLUE(P,C)
  & RMSIMAGE(P,D) & POSSADDOCLUE(C,I,1,2,3)
  >> EXISTSPM & ADDPROD(P,M,"MET" PL @ M @ "NOT R"),
  << (RED,"QUOTE PM," IMAGE,"QUOTE D"),
  << (CUE PM,"NCONC CN")
  & LMSTERM(P,M) @ ("NOT R") NEG(S) & LMSPREL(P,M)
  & RMSIMAGE(P,M,D) & RMSCLUE(P,M,"CUE PM,"NCONC CN)
  & REPROD(P,"MET" PL @ M @ "NOT R"),
  << (IMAGE,"QUOTE D"), << (CUE PM,"NCONC CN")
  & LMSTERM(P,M,POS,S) & LASTNEW(M) & NEGATE(ALL,0)
  & RMSCLUE(P,"CUE PM,"NCONC CN) & RMSIMAGE(P,I)
S2: "SPLIT PROD POS" = SPLITPROD(P,T,T,I,C,P,M)
  & NOT SATISFIES(TT,TT EQ NEG) & LMSTERM(P,M,N,S)
  & LMSPREL(P,M) & LASTNEW(M) & RMSCLUE(P,C)
  & RMSIMAGE(P,D) & POSSADDOCLUE(C,I,1,2,3)
  >> EXISTSPM & ADDPROD(P,M,"MET" PL @ M @ (TP) @ ("NOT TT"),
  << (RED,"QUOTE PM," IMAGE,"QUOTE D"),
  << (CUE PM,"NCONC CN")
  & LMSTERM(P,M,"NOT TT") NEG(S-1)
  & LMSPREL(P,M) @ M @ (TP)
  & RMSIMAGE(P,M,D) & RMSCLUE(P,M,"CUE PM,"NCONC CN)
  & REPROD(P,"MET" PL @ M @ (TP) @ (TT),
  << (RED,"QUOTE PM," IMAGE,"QUOTE D"),
  << (CUE PM,"NCONC CN")
  & LMSTERM(P,TT,POS,S-1) & LMSPREL(P,M) @ M @ (TP)
  & LASTNEW(M) & NEGATE(ALL)
  & RMSCLUE(P,"CUE PM,"NCONC CN) & RMSIMAGE(P,I)
S3: "SPLIT PREP 1" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 1) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,1) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,"XX,XX,XX,XX,XX,XX,XX,XX") & NEGATE(1)
S3A: "SPLIT PREP 1A" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 1) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,1) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,CODR CO,"XX,XX,XX,XX") & NEGATE(1)
S4: "SPLIT PREP 2" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 2) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,3) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,"XX,XX,XX,XX,XX,XX,XX,XX") & NEGATE(1)
S4A: "SPLIT PREP 2A" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 2) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,3) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,CODR CO,"XX,XX,XX,XX") & NEGATE(1)
S5: "SPLIT PREP 3" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 3) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,2) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,"XX,XX,XX,XX,XX,XX,XX,XX") & NEGATE(1)
S5A: "SPLIT PREP 3A" = SPLITPREP(P,T,T,I,S) & NOT EXTENDSTIMP
  & SATISFIES(S,S EQ 3) & STIMWORD(I,1,2,3)
  & RESPONSE(R,V,1,V2,V3) & VMEQL(V,2) & RMSCLUE(P,C)
  >> SPLITPROD(P,T,T,I,CODR CO,"XX,XX,XX,XX") & NEGATE(1)
S6: "SPLIT PREP 5" = SPLITPREP(P,T,T,I,S) & EXTENDSTIMP
  & RMSCLUE(P,C) & RESPONSE(R,V,1,V2,V3)
  & NOT SATISFIES(V,1,CODR CO) & NEGATE(1,2)
  >> SPLITPROD(P,T,T,I,"V,1,XX,XX,XX,XX,XX,XX") & NEGATE(1,2)
S7: "SPLIT PREP S2" = SPLITPREP(P,T,T,I,S) & EXTENDSTIMP
  & RMSCLUE(P,C) & RESPONSE(R,V,1,V2,V3)
  & SATISFIES(V,1,CODR CO) & NEGATE(1,2)
  >> SPLITPROD(P,T,T,I,"V,1,XX,XX,XX,XX,XX,XX")
  & NEGATE(1,2)
END: END.

```

Appendix B. Cross-Reference of EPAM Predicates

XREF OF EPAM PLUS NET AFTER THREE-PATH TEST

```

A
  LMSUSERS PN-3 -PN-6
  RMSUSERS PN-2
  ADDPROD
    RMSUSERS F1 S1 S2
  ADDOCLUE
    LMSUSERS C1 C2 C3
    RMSUSERS R4 -C1 -C2 -C3 S2
  COMPATNEG
    LMSUSERS R9 R9E
    NESTEDL R6
    RMSUSERS R8 -R9 R9E
  COMPATPOS
    LMSUSERS R6 R9E
    NESTEDL R9
    RMSUSERS -R6 R7A R7C -R9E
  COMPATPOSA
    LMSUSERS R6 R9
    RMSUSERS R3 -R6 -R9
  COMPATTEST
    LMSUSERS R7A R7C R8
    NESTEDL R6
    RMSUSERS R3 -R7 -R7A -R7C -R9
  CUE PROD
    LMSUSERS F3 R3 R4
    RMSUSERS F2 -R3 -R4
  CUE TRIPLE
    LMSUSERS F2 F3 R3 R5 C1 C2 C3
    RMSUSERS -F2 F3 -F3 R4 -R5 PM-1 PM-2 PM-3 PM-4 PM-5 PM-6
  EXTENDSARS
    LMSUSERS C4 C5 C6 C7 C8
    RMSUSERS R6 R9 -C4 -C5 -C6 -C7 -C8
  EXTENDSTIM
    LMSUSERS -S3 -S3A -S4 -S4A -S5 -S5A S6 S7
    RMSUSERS R9 -S6 -S7
  F
    LMSUSERS PN-1 -PN-4
  FIRED
    LMSUSERS F2 F3
    NESTEDL F1
    RMSUSERS -F2 -F3 PM-1 PM-1 PM-2 PM-3 PM-4 PM-5 PM-6
  IMAGE
    LMSUSERS F2 F3
    RMSUSERS -F2 -F3 PM-1 PM-1 PM-2 PM-3 PM-4 PM-5 PM-6
  LASTNEW
    LMSUSERS S1 S2
    RMSUSERS F1 S1 -S1 -S2 -S7
  LET1
    LMSUSERS F1 PM-1 PM-1 PM-2 PM-3 PM-4 PM-5 PM-6
    RMSUSERS F1 R5 -C1 PM-1 PM-1 PM-2
  LET2
    LMSUSERS E2 PM-3 PM-6
    RMSUSERS R5 -E2 PM-1 PM-2
  LET3
    LMSUSERS E3 PM-3 PM-5 PM-6
    RMSUSERS R5 -E3 PM-1 PM-2
  LMSPREL
    LMSUSERS S1 S2
    RMSUSERS F1 S1 S2 -S2
  LMSTERM
    LMSUSERS C4 C5 C6 C7 C8 S1 S2
    RMSUSERS F1 S1 -S1 -S2 -S2
  M
    LMSUSERS PN-3 -PN-5 PM-6
    RMSUSERS PM-1 PM-2
  N
    LMSUSERS PN-3 -PN-4 PM-5 PM-6
    RMSUSERS PM-1 PM-2
  OLDCLE TRIPLE
    LMSUSERS F3 R4
    RMSUSERS F2 -F3 -R4
  OLDEXTIM
    LMSUSERS R7A
    NESTEDL R6
    RMSUSERS R9
  OLDEREP
    LMSUSERS R10 R10A
    RMSUSERS F1 R5 -R10 -R10A

```

P
 LMSUSERS PM-2 -PM-4
 RMSUSERS PM-1
 POSSADDOUCUE
 LMSUSERS S1 S2
 RMSUSERS R6 R8 -S1 -S2
 R
 LMSUSERS PM-1 -PM-4
 REPLY
 LMSUSERS R1 R2 R3 R10
 NESTEDL R4 R10A
 RMSUSERS F3 R1 -R1 R2 -R2 -R3 R6
 REPLYPROO
 LMSUSERS R2 R3
 RMSUSERS F3 -R3
 REPPROO
 RMSUSERS S1 S2
 REPPEHSP
 RMSUSERS R2 C1 C2 C3
 RESPONSE
 LMSUSERS F1 R1 R2 R3 R4 R5 S3 S3A S4 S4A S5 S5A S6 S7
 RMSUSERS -F1 -R1 -R2 -R3 R10 R10A
 RESTIM
 LMSUSERS R5
 RMSUSERS -R5 S5A R5
 RESTIMHOLD
 LMSUSERS R5A
 RMSUSERS R4 -R5A
 RMSUCUE
 LMSUSERS R2 C1 C2 C3 S1 S2 S3 S3A S4 S4A S5 S5A S6 S7
 RMSUSERS F1 C1 -C1 C2 -C2 C3 -C3 S1 -S1 S2 -S2
 RMSIMAGE
 LMSUSERS C1 C2 C3 S1 S2
 RMSUSERS F1 R2 -R2 S1 -S1 S2 -S2
 SPLITPREP
 LMSUSERS S3 S3A S4 S4A S5 S5A S6 S7
 RMSUSERS C4 C5 C6 C7 C8 -S3 -S3A -S4 -S4A -S5 -S5A -S6 -S7
 SPLITPROO
 LMSUSERS S1 S2
 RMSUSERS -S1 -S2 S3 S3A S4 S4A S5 S5A S6 S7
 STIMCHK
 LMSUSERS F1 F2
 RMSUSERS F1 -F2 R5
 STIMREM
 LMSUSERS R5 E1 E2 E3 E4
 NESTEDL E5
 RMSUSERS F2 F3 R4 -R5 -E4
 STIMREMREM
 LMSUSERS E4 E5
 RMSUSERS R5 -E4 -E5
 STIMWORD
 LMSUSERS F1 R9 S3 S3A S4 S4A S5 S5A
 U
 RMSUSERS PM-1

Appendix C. Detailed Trace on Three-Pair Test

FIRST SEG OF THREE-PAIR TEST

TOP LEVEL ASSERT (STIM FIB)
 INSERTING (LETS L1-1) (F L1-1) (LET2 L2-1) (I L2-1) (LETS L3-1) (B L3-1)
 (STIMCHK L1-1 L2-1 L3-1) (STIMWORD F 1 B) E1E2E3F2F1S6A5S4A6S3A5S2A5

7
 TOP LEVEL ASSERT (RESP NUM)
 INSERTING (RESPONSE NUM N U M) F1/

1. F1-1 "INIT NET"
 USING (STIMCHK L1-1 L2-1 L3-1) (STIMWORD F 1 B) (RESPONSE NUM N U M)
 ADOPROO
 PM-1 "NET"
 LMS (LETS V1) (F V1)
 RMS (EXISTS C1) (FIRED (QUOTE PM-1)) (IMAGE (QUOTE DUMD)) (LETS C1) (M C1)
 (CUETRIPLE (QUOTE PM-1) C1 (QUOTE XX) (QUOTE XX))
 VARS V1
 ADOPROO
 PM-1 "NET"
 LMS (LETS V1) (NOT (F V1))
 RMS (FIRED (QUOTE PM-1)) (IMAGE (QUOTE DUMD))
 (CUETRIPLE (QUOTE PM-1) (QUOTE XX) (QUOTE XX) (QUOTE XX))
 VARS V1
 WARNING (L1-1) ALREADY UNDER LETS "A"
 WARNING (L1-1 L2-1 L3-1) ALREADY UNDER STIMCHK "A"
 INSERTING (LETS L1-1) (STIMCHK L1-1 L2-1 L3-1) (ADOPROO PM-1)
 (LMSPEL PM-1 (LETS V1)) (LMSTERM PM-1 (F V1)) POS 1) (RMSIMAGE PM-1 DUMD)
 (RMSUCUE PM-1 (CUE PM-1 N XX XX)) (ADOPROO PM-1) (LMSPEL PM-1 (LETS V1))
 (LMSTERM PM-1 (NOT (F V1))) NEG 1) (RMSIMAGE PM-1 DUMD)
 (RMSUCUE PM-1 (CUE PM-1 XX XX XX)) (LASTIM PM-1) (OLDRESP NUM N U M)
 (NOT (RESPONSE NUM N U M)) PM-1/PM-1/

2. PM-1-1 "NET"
 USING (LETS L1-1) (F L1-1)
 INSERTING (FIRED PM-1) (IMAGE DUMD) (LETS C1-1) (M C1-1)
 (CUETRIPLE PM-1 C1-1 XX XX) F2F2/

3. F2-1 "STIM F1PE"
 USING (FIRED PM-1) (STIMCHK L1-1 L2-1 L3-1) (IMAGE DUMD)
 (CUETRIPLE PM-1 C1-1 XX XX)
 INSERTING (STIMREM L1-1 L2-1 L3-1 STIM) (OLDCUETRIPLE C1-1 XX XX) (CUEPROD PM-1)
 (NOT (FIRED PM-1)) (NOT (STIMCHK L1-1 L2-1 L3-1)) (NOT (IMAGE DUMD))
 (NOT (CUETRIPLE PM-1 C1-1 XX XX)) RSE1/

4. E1-1 "STIM REM1"
 USING (STIMREM L1-1 L2-1 L3-1 STIM) (LETS L1-1)
 INSERTING (NOT (LETS L1-1)) E4E3/

5. E3-1 "STIM REM3"
 USING (STIMREM L1-1 L2-1 L3-1 STIM) (LETS L3-1)
 INSERTING (NOT (LETS L3-1)) E2/

6. E2-1 "STIM REM2"
 USING (STIMREM L1-1 L2-1 L3-1 STIM) (LET2 L2-1)
 INSERTING (NOT (LET2 L2-1)) R4F2R3PM-1/PM-1/

7. PM-1-1 "NET"
 USING (LETS C1-1)
 INSERTING (FIRED PM-1) (IMAGE DUMD) (CUETRIPLE PM-1 XX XX XX) F2F2/

8. F3-1 "CUE F1PE"
 USING (FIRED PM-1) (CUEPROD PM-1) (OLDCUETRIPLE C1-1 XX XX)
 (CUETRIPLE PM-1 XX XX XX) (IMAGE DUMD)
 INSERTING (STIMREM XX XX XX DUM) (STIMREM C1-1 XX XX CUE) (REPLY DUMD)
 (REPLYPROD PM-1) (CUETRIPLE PM-1 C1-1 XX XX) (NOT (FIRED PM-1))
 (NOT (OLDCUETRIPLE C1-1 XX XX)) (NOT (CUETRIPLE PM-1 XX XX XX))
 (NOT (IMAGE DUMD)) E1/

9. E1-2 "STIM REM1"
 USING (STIMREM C1-1 XX XX CUE) (LETS C1-1)
 INSERTING (NOT (LETS C1-1)) R5E2E3E4R2R3R10/

10. R10-1 "REM RESP"
 USING (REPLY DUMD) (OLDRESP NUM N U M)
 INSERTING (RESPONSE NUM N U M) (NOT (OLDRESP NUM N U M)) R1/R2/

1 11. P2-1 "REPLY MALL"
 USING (RESPONSE NUM N U M) (REPLY DUARD) (REPLYPROD PH-1)
 (PHSCUE PH-1 (CUE PH-1 XX XX XX))
 REPPHS
 PH-1 "NET"
 LMS (LE1 V1) (NOT (P V1))
 RMS (FIRED (QUOTE PH-1)) (IMAGE (QUOTE NUM))
 (CUE TRIPLE (QUOTE PH-1) (QUOTE X1) (QUOTE X2) (QUOTE X3))
 UMS V1
 INSERTING (REPLY OK) (REPPHS PH-1) (PHSCUE PH-1 NUM)
 (NOT (PHSCUE PH-1 DUARD)) (NOT (RESPONSE NUM N U M)) (NOT (REPLY DUARD))
 R3R10R2R1C1C2C3S1S2S3S4S5S6S7S8S9S10S11S12S13S14S15S16S17S18S19S20S21S22S23S24S25S26S27S28S29S30S31S32S33S34S35S36S37S38S39S40S41S42S43S44S45S46S47S48S49S50S51S52S53S54S55S56S57S58S59S60S61S62S63S64S65S66S67S68S69S70S71S72S73S74S75S76S77S78S79S80S81S82S83S84S85S86S87S88S89S90S91S92S93S94S95S96S97S98S99S100

ADDPROD (PH-1) (PH-1)
 REPPHS (PH-1)

REPLY (OK)

SAVED PH-1 PH-1

NIL

TOP LEVEL ASSERT (STIM RAM)
 INSERTING (LE1 L1-1) (R L1-1) (LE2 L2-1) (A L2-1) (LE3 L3-1) (M L3-1)
 (STIMORD L1-1 L2-1 L3-1) (STIMORD R A M) E1PH-1/

1 12. PH-1-2 "NET"
 USING (LE1 L1-1)
 INSERTING (FIRED PH-1) (IMAGE NUM) (CUE TRIPLE PH-1 XX XX XX) F3F2/

1 13. F2-2 "STIM FIPE"
 USING (FIRED PH-1) (STIMORD L1-1 L2-1 L3-1) (IMAGE NUM)
 (CUE TRIPLE PH-1 XX XX XX)
 INSERTING (STIMORD L1-1 L2-1 L3-1 STIM) (CUE TRIPLE PH-1 XX XX XX) (CUEPROD PH-1)
 (NOT (FIRED PH-1)) (NOT (STIMORD L1-1 L2-1 L3-1)) (NOT (IMAGE NUM))
 (NOT (CUE TRIPLE PH-1 XX XX XX)) E3/

1 14. E3-2 "STIM PEH3"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE3 L3-1)
 INSERTING (NOT (LE3 L3-1)) E2/

1 15. E2-2 "STIM PEH2"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE2 L2-1)
 INSERTING (NOT (LE2 L2-1)) R5E1/

1 16. E1-3 "STIM PEH1"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE1 L1-1)
 INSERTING (NOT (LE1 L1-1)) E4F3F2F1S3S4S5S6S7S8S9S10S11S12S13S14S15S16S17S18S19S20S21S22S23S24S25S26S27S28S29S30S31S32S33S34S35S36S37S38S39S40S41S42S43S44S45S46S47S48S49S50S51S52S53S54S55S56S57S58S59S60S61S62S63S64S65S66S67S68S69S70S71S72S73S74S75S76S77S78S79S80S81S82S83S84S85S86S87S88S89S90S91S92S93S94S95S96S97S98S99S100

TOP LEVEL ASSERT (RESP POP)
 INSERTING (RESPONSE POP P O R) R4/

1 17. R4-1 "REPLY ABS"
 USING (RESPONSE POP P O R) (CUE TRIPLE PH-1 XX XX XX) (CUEPROD PH-1)
 INSERTING (STIMORD XX XX XX CUE) (ADDUCUE PH-1 P O R) (CUE TRIPLE PH-1 XX XX XX)
 (PESTIMORD OK) (NOT (CUE TRIPLE PH-1 XX XX XX)) (NOT (CUEPROD PH-1)) R5E1E2
 E3E4C1/

1 18. C1-1 "CM CUE"
 USING (ADDUCUE PH-1 P O R) (CUE TRIPLE PH-1 XX XX XX)
 (PHSCUE PH-1 (CUE PH-1 XX XX XX)) (PHSCUE PH-1 NUM)
 REPPHS
 PH-1 "NET"
 LMS (LE1 V1) (NOT (P V1))
 RMS (EXISTS C1) (FIRED (QUOTE PH-1)) (IMAGE (QUOTE NUM)) (LE1 C1) (P C1)
 (CUE TRIPLE (QUOTE PH-1) C1 (QUOTE X1) (QUOTE X2))
 UMS V1
 INSERTING (REPPHS PH-1) (PHSCUE PH-1 (CUE PH-1 P XX XX))
 (NOT (ADDUCUE PH-1 P O R)) (NOT (PHSCUE PH-1 (CUE PH-1 XX XX XX))) C2C3S1
 S2P2S3S4S5S6S7S8S9S10S11S12S13S14S15S16S17S18S19S20S21S22S23S24S25S26S27S28S29S30S31S32S33S34S35S36S37S38S39S40S41S42S43S44S45S46S47S48S49S50S51S52S53S54S55S56S57S58S59S60S61S62S63S64S65S66S67S68S69S70S71S72S73S74S75S76S77S78S79S80S81S82S83S84S85S86S87S88S89S90S91S92S93S94S95S96S97S98S99S100

1 19. R5A-1 "PE STIM"
 USING (PESTIMORD OK)
 INSERTING (PESTIM OK) (NOT (PESTIMORD OK)) R5/

1 20. R5-1 "RE STIM"
 USING (PESTIM OK) (STIMORD L1-1 L2-1 L3-1 STIM) (CUE TRIPLE PH-1 XX XX XX)
 (RESPONSE POP P O R)
 INSERTING (STIMORD OK) (LE1 L1-1) (LE2 L2-1) (LE3 L3-1)
 (STIMORD L1-1 L2-1 L3-1) (CUEPROD PH-1 P O R) (NOT (PESTIM OK))
 (NOT (STIMORD L1-1 L2-1 L3-1 STIM)) (NOT (CUE TRIPLE PH-1 XX XX XX))
 (NOT (RESPONSE POP P O R)) E5E4/

1 21. E4-1 "STIM PEH PEH"
 USING (STIMORD OK) (STIMORD XX XX XX CUE)
 INSERTING (NOT (STIMORD OK)) (NOT (STIMORD XX XX XX CUE)) PH-1E1PH-1/

1 22. PH-1-3 "NET"
 USING (LE1 L1-1)
 INSERTING (FIRED PH-1) (IMAGE NUM) (LE1 C1-1) (P C1-1)
 (CUE TRIPLE PH-1 C1-1 XX XX) F2/

1 23. F2-3 "STIM FIPE"
 USING (FIRED PH-1) (STIMORD L1-1 L2-1 L3-1) (IMAGE NUM)
 (CUE TRIPLE PH-1 C1-1 XX XX)
 INSERTING (STIMORD L1-1 L2-1 L3-1 STIM) (CUE TRIPLE PH-1 C1-1 XX XX) (CUE
 (NOT (FIRED PH-1)) (NOT (STIMORD L1-1 L2-1 L3-1)) (NOT (IMAGE NUM))
 (NOT (CUE TRIPLE PH-1 C1-1 XX XX)) E4E3/

1 24. E3-3 "STIM PEH3"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE3 L3-1)
 INSERTING (NOT (LE3 L3-1)) E2/

1 25. E2-3 "STIM PEH2"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE2 L2-1)
 INSERTING (NOT (LE2 L2-1)) E1/

1 26. E1-4 "STIM PEH1"
 USING (STIMORD L1-1 L2-1 L3-1 STIM) (LE1 L1-1)
 INSERTING (NOT (LE1 L1-1)) R5R4F3R3PH-1PH-1/

1 27. PH-1-4 "NET"
 USING (LE1 C1-1)
 INSERTING (FIRED PH-1) (IMAGE NUM) (LE1 C1-2) (P C1-2)
 (CUE TRIPLE PH-1 C1-2 XX XX) F2F3/

1 28. F3-2 "CUE FIPE"
 USING (FIRED PH-1) (CUEPROD PH-1) (CUE TRIPLE PH-1 C1-1 XX XX)
 (CUE TRIPLE PH-1 C1-2 XX XX) (IMAGE NUM)
 INSERTING (STIMORD C1-2 XX XX DUM) (STIMORD C1-1 XX XX CUE) (REPLY NUM)
 (CUEPROD PH-1) (CUE TRIPLE PH-1 C1-1 XX XX) (NOT (FIRED PH-1))
 (NOT (CUE TRIPLE PH-1 C1-1 XX XX)) (NOT (CUE TRIPLE PH-1 C1-2 XX XX))
 (NOT (IMAGE NUM)) E4E3E2C1/

1 29. E1-5 "STIM PEH1"
 USING (STIMORD C1-1 XX XX CUE) (LE1 C1-1)
 INSERTING (NOT (LE1 C1-1))

1 30. E1-6 "STIM PEH1"
 USING (STIMORD C1-2 XX XX DUM) (LE1 C1-2)
 INSERTING (NOT (LE1 C1-2)) R5R3R10/

1 31. R1R-2 "PEH RESP"
 USING (REPLY NUM) (CUEPROD PH-1 P O R)
 INSERTING (RESPONSE POP P O R) (NOT (CUEPROD PH-1 P O R)) R2/R1/R3/

1 32. R3-1 "REPLY WRONG"
 USING (RESPONSE POP P O R) (REPLY NUM) (CUEPROD PH-1) (REPLYPROD PH-1)
 (CUE TRIPLE PH-1 C1-1 XX XX)
 INSERTING (COMPATTEST P N C1-1) (COMPATTEST R N XX) (COMPATTEST O U XX)
 (COMPATTEST PH-1 POR P O R PH-1) (NOT (REPLY NUM)) (NOT (CUEPROD PH-1))
 (NOT (REPLYPROD PH-1)) R8/

1 33. R8-1 "COMPAT T-"
 USING (COMPATTEST P N C1-1)
 INSERTING (COMPATNEG C1-1) (NOT (COMPATTEST P N C1-1)) R8R8/

1 34. R9-1 "COMPAT -"
 USING (COMPATNEG C1-1) (COMPATTEST PH-1 POR P O R PH-1) (STIMORD R A M)
 INSERTING (EXTENDS PH-1 DUARD R A M) (EXTENDS PH-1) (PESTIM OK)
 (CUEPROD PH-1) (COMPATTEST PH-1 POR P O R PH-1) (NOT (COMPATNEG C1-1))
 (NOT (COMPATTEST PH-1 POR P O R PH-1)) C4/

1 35. C4-1 "EXT LS NEG"
 USING (EXTENDS PH-1 DUARD R A M) (LMSTERN PH-1) (NOT (P V1)) NEG 1)
 INSERTING (SPLITTER PH-1 (P V1) NEG DUARD 1)
 (NOT (EXTENDS PH-1 DUARD R A M)) S3/S3A/S4/S4A/S5/S5A/S6/S6A/S7/

111-52

SAVED PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT (STIM PEK)

INSERTING (LETS L1-1) (P L1-1) (LET2 L2-1) (E L2-1) (LETS L3-1) (K L3-1)
(STIMCHK L1-1 L2-1 L3-1) (STIMORD P E K) PN-1/PN-1PN-2/

1 58. PN-2-2 "NET"

USING (LETS L1-1) (P L1-1)

INSERTING (FIPOD PN-2) (IMAGE POR) (CUETRIPLE PN-2 XX XX XX) F2F2/

1 59. F2-5 "STIM FIPE"

USING (FIPOD PN-2) (STIMCHK L1-1 L2-1 L3-1) (IMAGE POR)

(CUETRIPLE PN-2 XX XX XX)

INSERTING (STIMPEM L1-1 L2-1 L3-1 STIM) (OLDUETRIPLE XX XX XX) (CUEPROD PN-2)
(NOT (FIPOD PN-2)) (NOT (STIMCHK L1-1 L2-1 L3-1)) (NOT (IMAGE POR))
(NOT (CUETRIPLE PN-2 XX XX XX)) E2/

1 60. E2-5 "STIM PEM2"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LETS L2-1)

INSERTING (NOT (LETS L2-1)) E3/

1 61. E3-5 "STIM PEM3"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LETS L3-1)

INSERTING (NOT (LETS L3-1)) R5E1/

1 62. E1-9 "STIM PEM1"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LETS L1-1)

INSERTING (NOT (LETS L1-1)) E4P4F3R3F1P55S4S5S4S5S4S5S4S5

T

TOP LEVEL ASSERT (RESP NMI)

INSERTING (RESPONSE NMI N A M) S7S6R2P5F1R3P1P4/

1 63. R4-2 "PEPLY ABS"

USING (RESPONSE NMI N A M) (OLDUETRIPLE XX XX XX) (CUEPROD PN-2)

INSERTING (STIMPEM XX XX XX CUE) (ADDTOCUE PN-2 N A M) (CUETRIPLE PN-2 XX XX XX)
(RESTIMHOLD OK) (NOT (OLDUETRIPLE XX XX XX)) (NOT (CUEPROD PN-2)) E4P5E2
E3E1C3/C2/C1/

1 64. C1-2 "CH CUE"

USING (ADDTOCUE PN-2 N A M) (CUETRIPLE PN-2 XX XX XX)

(RHS CUE PN-2 (CUE PN-2 XX XX XX)) (RHS IMAGE PN-2 POR)

REPHAS

PN-2 "NET"

LMS (LETS V1) (P V1)

RHS (EXISTS C1) (FIPOD (QUOTE PN-2)) (IMAGE (QUOTE POR)) (LETS C1) (N C1)

(CUETRIPLE (QUOTE PN-2) C1 (QUOTE XX) (QUOTE XX))

VAR5 V1

INSERTING (REPPHAS PN-2) (PHSCUE PN-2 (CUE PN-2 N XX XX))

(NOT (ADDTOCUE PN-2 N A M)) (NOT (PHSCUE PN-2 (CUE PN-2 XX XX XX))) C1C3C2

S2S1P253S3S4S5S4S5S5S4S5S6S7F2F3P3P5A/

1 65. PSA-2 "PE STIM"

USING (PESTIMHOLD OK)

INSERTING (PESTIM OK) (NOT (PESTIMHOLD OK)) R5/

1 66. PS-3 "PE STIM"

USING (PESTIM OK) (STIMPEM L1-1 L2-1 L3-1 STIM) (CUETRIPLE PN-2 XX XX XX)

(RESPONSE NMI N A M)

INSERTING (STIMPEM OK) (LETS L1-1) (LET2 L2-1) (LETS L3-1)

(STIMCHK L1-1 L2-1 L3-1) (OLDRESP NMI N A M) (NOT (PESTIM OK))

(NOT (STIMPEM L1-1 L2-1 L3-1 STIM)) (NOT (CUETRIPLE PN-2 XX XX XX))

(NOT (RESPONSE NMI N A M)) E4/

1 67. E4-4 "STIM PEM PEM"

USING (STIMPEM OK) (STIMPEM XX XX XX CUE)

INSERTING (NOT (STIMPEM OK)) (NOT (STIMPEM XX XX XX CUE)) PN-3/PN-1PN-1PN-2/

1 68. PN-2-3 "NET"

USING (LETS L1-1) (P L1-1)

INSERTING (FIPOD PN-2) (IMAGE POR) (LETS C1-1) (N C1-1)

(CUETRIPLE PN-2 C1-1 XX XX) F2/

1 69. F2-6 "STIM FIPE"

USING (FIPOD PN-2) (STIMCHK L1-1 L2-1 L3-1) (IMAGE POR)

(CUETRIPLE PN-2 C1-1 XX XX)

INSERTING (STIMPEM L1-1 L2-1 L3-1 STIM) (OLDUETRIPLE C1-1 XX XX) (CUEPROD PN-2)

(NOT (FIPOD PN-2)) (NOT (STIMCHK L1-1 L2-1 L3-1)) (NOT (IMAGE POR))

(NOT (CUETRIPLE PN-2 C1-1 XX XX)) E1/

1 70. E1-10 "STIM PEM1"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LETS L1-1)

INSERTING (NOT (LETS L1-1)) E3/

1 71. E3-6 "STIM PEM3"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LETS L3-1)

INSERTING (NOT (LETS L3-1)) E2/

1 72. E2-6 "STIM PEM2"

USING (STIMPEM L1-1 L2-1 L3-1 STIM) (LET2 L2-1)

INSERTING (NOT (LET2 L2-1)) R5E1P4F3R3F1P55S4S5S4S5S4S5S4S5S4S5

1 73. PN-3-1 "NET"

USING (LETS C1-1)

INSERTING (FIPOD PN-3) (IMAGE NMI) (CUETRIPLE PN-3 XX XX XX) F2F3/

1 74. F3-4 "CUE FIPE"

USING (FIPOD PN-3) (CUEPROD PN-2) (OLDUETRIPLE C1-1 XX XX)

(CUETRIPLE PN-3 XX XX XX) (IMAGE NMI)

INSERTING (STIMPEM XX XX XX CUE) (STIMPEM C1-1 XX XX CUE) (REPLY NMI)

(CUEPROD PN-3) (CUETRIPLE PN-2 C1-1 XX XX) (NOT (FIPOD PN-3))

(NOT (OLDUETRIPLE C1-1 XX XX)) (NOT (CUETRIPLE PN-3 XX XX XX))

(NOT (IMAGE NMI)) E1E1/

1 75. E1-11 "STIM PEM1"

USING (STIMPEM C1-1 XX XX CUE) (LETS C1-1)

INSERTING (NOT (LETS C1-1)) E3E2R5R10/

1 76. R1R-4 "PEM RESP"

USING (REPLY NMI) (OLDRESP NMI N A M)

INSERTING (RESPONSE NMI N A M) (NOT (OLDRESP NMI N A M)) R3/

1 77. R3-3 "PEPLY WRONG"

USING (RESPONSE NMI N A M) (REPLY NMI) (CUEPROD PN-2) (REPLYPROD PN-3)

(CUETRIPLE PN-2 C1-1 XX XX)

INSERTING (COMPATTEST N N C1-1) (COMPATTEST M M XX) (COMPATTEST A U XX)

(COMPATPOSA PN-3 NMI N A M PN-2) (NOT (REPLY NMI)) (NOT (CUEPROD PN-2))

(NOT (REPLYPROD PN-3)) R8/R7C/

1 78. R7C-1 "COMPAT T OK"

USING (COMPATTEST N N C1-1)

INSERTING (COMPATPOS C1-1) (NOT (COMPATTEST N N C1-1)) R6/R5R7/

1 79. R7-3 "COMPAT T"

USING (COMPATTEST A U XX)

INSERTING (COMPATPOS XX) (NOT (COMPATTEST A U XX))

1 80. R7-4 "COMPAT T"

USING (COMPATTEST M M XX)

WARNING (XX) ALREADY UNDER COMPATPOS

INSERTING (COMPATPOS XX) (NOT (COMPATTEST M M XX)) R6/

1 81. R6-2 "COMPAT A"

USING (COMPATPOS XX) (COMPATPOSA PN-3 NMI N A M PN-2)

INSERTING (EXTENDLSPS PN-3 NMI N A M) (POSSADOTOCUE PN-2 N A M) (REPLY OK)

(NOT (COMPATPOS XX)) (NOT (COMPATPOSA PN-3 NMI N A M PN-2)) C5/C6/C7/C8/C4/

1 82. C4-3 "EXT LS NEG"

USING (EXTENDLSPS PN-3 NMI N A M)

(LASTPEM PN-3) (NOT (P V1)) (NOT (R V1)) (NOT (P V1)) NEG 1)

INSERTING (SPLITPEP PN-3 (N V1) NEG NMI 1) (NOT (EXTENDLSPS PN-3 NMI N A M))

S7S3P54/S4A/S5/S5A/S6S3/

1 83. S3-2 "SPLIT PREP 1"

USING (SPLITPEP PN-3 (N V1) NEG NMI 1) (STIMORD P E K) (RESPONSE NMI N A M)

(PHSCUE PN-3 (CUE PN-3 XX XX XX))

INSERTING (SPLITPROD PN-3 (N V1) NEG NMI (XX XX XX) (XX XX XX))

(NOT (SPLITPEP PN-3 (N V1) NEG NMI 1)) S1/

1 84. S1-3 "SPLIT PROD NEG"

USING (SPLITPROD PN-3 (N V1) NEG NMI (XX XX XX) (XX XX XX))

(LASTPEM PN-3) (NOT (P V1)) (NOT (R V1)) (NOT (P V1)) NEG 1)

(LASTPEP PN-3 (LETS V1)) (LASTPEM PN-3) (PHSCUE PN-3 (CUE PN-3 XX XX XX))

(PHSIMAGE PN-3 NMI) (POSSADOTOCUE PN-2 N A M)

ADOPROD

PN-4 "NET"

```

LMS (LET1 V1) (NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (N V1))
RHS (FIPED (QUOTE PN-4)) (IMAGE (QUOTE NUM))
(CUETRIPL (QUOTE PN-4) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS V1
REPPROD
PN-3 * NET *
LMS (LET1 V1) (N V1)
RHS (FIPED (QUOTE PN-3)) (IMAGE (QUOTE NUM))
(CUETRIPL (QUOTE PN-3) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS V1
INSERTING (ADOPPOD PN-4)
(LMSTERM PN-4 ((NOT (F V1)) (NOT (P V1)) (NOT (P V1)) (NOT (N V1))) NEG 1)
(LMSPEL PN-4 ((LET1 V1)) (RHSIMAGE PN-4 NUM))
(RHSCUE PN-4 (CUE PN-4 XX XX XX)) (PEPPROD PN-3)
(LMSTERM PN-3 ((N V1)) POS 1) (LASTNEW PN-4)
(NOT (SPLITPOD PN-3 (N V1) NEG NUM (XX XX XX) (XX XX XX)))
(NOT (LMSTERM PN-3 ((NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (N V1))) NEG 1))
(NOT (LASTNEW PN-3)) (NOT (RHSCUE PN-3 (CUE PN-3 XX XX XX)))
(NOT (RHSIMAGE PN-3 NUM)) (NOT (POSSADOCUE PN-2 N A M))
(RHSCUE PN-3 (CUE PN-3 XX XX XX)) (RHSIMAGE PN-3 NUM) S251C8C7C6C5C4C1C2C3R2
S3S3A54S4S5S5A5S6S7R1/R3R1D4R5F1F2F3

```

```

ADOPPOD (PN-4)
REPPROD (PN-3)
REPRHSP (PN-2)

```

REPLY (OK)

SAVED PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT NIL

```

PN-1 * NET *
LMS (LET1 V1) (F V1)
RHS (EXISTS C1) (FIPED (QUOTE PN-1)) (IMAGE (QUOTE DUNO)) (LET1 C1) (N C1)
(CUETRIPL (QUOTE PN-1) C1 (QUOTE XX) (QUOTE XX))
VARS V1
PN-1 * NET *
LMS (LET1 V1) (R V1)
RHS (EXISTS C1) (FIPED (QUOTE PN-1)) (IMAGE (QUOTE DUNO)) (LET1 C1) (P C1)
(CUETRIPL (QUOTE PN-1) C1 (QUOTE XX) (QUOTE XX))
VARS V1
PN-2 * NET *
LMS (LET1 V1) (P V1)
RHS (EXISTS C1) (FIPED (QUOTE PN-2)) (IMAGE (QUOTE POR)) (LET1 C1) (N C1)
(CUETRIPL (QUOTE PN-2) C1 (QUOTE XX) (QUOTE XX))
VARS V1
PN-3 * NET *
LMS (LET1 V1) (N V1)
RHS (FIPED (QUOTE PN-3)) (IMAGE (QUOTE NUM))
(CUETRIPL (QUOTE PN-3) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS V1
PN-4 * NET *
LMS (LET1 V1) (NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (N V1))
RHS (FIPED (QUOTE PN-4)) (IMAGE (QUOTE NUM))
(CUETRIPL (QUOTE PN-4) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS V1

```

```

LASTNEW (PN-4)
LMSPEL (PN-1 ((LET1 V1)) (PN-1 ((LET1 V1)) (PN-2 ((LET1 V1))
(PN-3 ((LET1 V1)) (PN-4 ((LET1 V1))
LMSTERM (PN-1 ((R V1)) POS 1) (PN-1 ((F V1)) POS 1) (PN-2 ((P V1)) POS 1)
(PN-3 ((N V1)) POS 1)
(PN-4 ((NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (N V1))) NEG 1)
RHSCUE (PN-1 (CUE PN-1 P XX XX)) (PN-1 (CUE PN-1 N XX XX))
(PN-2 (CUE PN-2 N XX XX)) (PN-3 (CUE PN-3 XX XX XX))
(PN-4 (CUE PN-4 XX XX XX))
RHSIMAGE (PN-1 DUNO) (PN-1 DUNO) (PN-2 POR) (PN-3 NUM) (PN-4 NUM)

```

RUN TIME 44.3 SEC

EXAM	TPY	FILE	MMCT	E/F	E/T	T/F
376	122	04	372	4.48	3.08	1.45
0.110	0.363	0.520	0.119	SEC AVG		

224 INSERTS 140 DELETES 7 WARNINGS 5 NEW OBJECTS
MAX SMPX LENGTH 33

CORE (FREE,FULL) (0640 . 1013) USED (1302 . 109)

FACTS LOADS (EPAM . EXP) (EPAPER . MAC) EPAM RUN SMPXEMPTY SMPXEMPTY SMPXEMPTY
SMPXEMPTY SMPXEMPTY SMPXEMPTY

TRACE

(F1-1 PN-1-1 F2-1 E1-1 E3-1 E2-1 PN-1-1 F3-1 E1-2 R10-1 R2-1 PN-1-2 F2-2 E3-2
E2-2 E1-3 R4-1 C1-1 PSA-1 R5-1 E4-1 PN-1-3 F2-3 E3-3 E2-3 E1-4 PN-1-4 F3-2
E1-5 E1-6 P10-2 R3-1 P0-1 R9-1 C4-1 S7-1 S1-1 R5-2 E4-2 E4-3 PN-1-5 F2-4 E2-4
E3-4 E1-7 PN-2-1 F3-3 E1-8 R10-3 R3-2 R7-1 R7-2 R70-1 R5-1 C4-2 S3-1 S1-2
PN-2-2 F2-5 E2-5 E3-5 E1-9 R4-2 C1-2 PSA-2 R5-3 E4-4 PN-2-3 F2-6 E1-10 E3-6
E2-6 PN-3-1 F3-4 E1-11 R10-4 R3-3 R7C-1 R7-3 R7-4 R5-2 C4-3 S3-2 S1-3)

FIRE 20 OUT OF 46 PRODS

(SWAPS (EPAM))

CLOSED (EPAM) . EXP)

(CLOSED (EPAM) . DBS))

SECOND SEG OF THREE PAIRS

TOP LEVEL ASSERT (STIM FIB)	PN-2PN-1/
1. PN-1-1 "NET" F3F2/	
2. F2-1 "STIM F1P" R5E2/	
3. E2-1 "STIM PEN2" E3/	
4. E3-1 "STIM MEM3" E1/	
5. E1-1 "STIM MEM1" E4R4F3R3PN-3/	
6. PN-3-1 "NET" F2F3/	
7. F3-1 "CUE F1P" E4E2E3E1/	
8. E1-2 "STIM MEM1" R5R2R3R1R1C1C2C3F2F3F1S3S3A54S4S5S5A5R9	

REPLY (NUM)

TOP LEVEL ASSERT (RESP NUM)	PSR2/R3/
9. R3-1 "REPLY WONG" R6/R7R7C/	
10. R7C-1 "COMPAT 1 OK" R6/R5E87/	
11. R7-1 "COMPAT 1"	
12. R7-2 "COMPAT 1"	== R6/
13. R6-1 "COMPAT 1"	C0/C7/
14. C7-1 "EXT LS POS 3"	S6S5A/S5/S4A/S4/
15. S4-1 "SPLIT PREP 2"	S1/S2/
16. S2-1 "SPLIT PROD POS"	C2/
17. C2-1 "CM CUE3"	S2C3C1S1C2S7S3A5R2S4S4S5S5A5C6C7C6C5C4R1R1/

```

ADOPPOD (PN-5)
REPPROD (PN-3)
REPRHSP (PN-1)

```

REPLY (OK)

SAVED PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT (STIM NUM)	PN-3PN-4/PN-2PN-1E(PN-1/
18. PN-1-1 "NET" F3F2/	
19. F2-2 "STIM F1P" R5E3/	
20. E3-2 "STIM MEM3" E2/	
21. E2-2 "STIM MEM2" E4E1/	
22. E1-3 "STIM MEM1" R4F3R3PN-3PN-1PN-2/	
23. PN-2-1 "NET" F2F3/	
24. F3-2 "CUE F1P" E4E3E2E1/	
25. E1-4 "STIM MEM1"	
26. E1-5 "STIM MEM1"	R5R2R3R1R1F2C3C2C1F3PN-3PN-4SF1R5S5A5S5A5R9

REPLY (POR)

T
TOP LEVEL ASSERT (RESP POR) S3S3AS6S7R2/RSR3/RMF1R1/
! 27. R1-1 "REPLY MATCH" R1R1OP3R2

REPLY (OK)

SAVED PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT (STIM PEK) PN-1E1PN-2/
! 28. PN-2-2 "NET" F3F2/
! 29. F2-3 "STIM F1PE" P5E2/
! 30. E2-3 "STIM PEH2" E3/
! 31. E3-3 "STIM PEH3" E4E1/
! 32. E1-6 "STIM PEH1" R4F3R3PN-1PN-3PN-4/PN-5PN-2/PN-1F1S3S3AS4S4AS5
SSARS

T
TOP LEVEL ASSERT (RESP NMI) S4S4AS5S5AF1R1/
! 33. P4-1 "REPLY ABS" P5E1/
! 34. E1-7 "STIM PEH1" E3E2E4C1/C3/C2/
! 35. C2-2 "CH CUE3" S1S2C3C1C2S7S3AS3R2S4S4AS5S5AS6F3R3F2R5A/
! 36. P5A-1 "PE STIM-" P5/
! 37. P5-1 "PE STIM" E4/
! 38. E4-1 "STIM PEH REM" PN-5/PN-4/PN-3PN-1PN-2/
! 39. PN-2-3 "NET" F2/
! 40. F2-4 "STIM F1PE" E2/
! 41. E2-4 "STIM PEH2" E3/
! 42. E3-4 "STIM PEH3" P5E1/
! 43. E1-8 "STIM PEH1" E4R4F3R3PN-3/
! 44. PN-3-2 "NET" F2F3/
! 45. F3-3 "CUE F1PE" E4E2E1/
! 46. E1-9 "STIM PEH1" E3/
! 47. E3-5 "STIM PEH3" P5P2P3P10/
! 48. P10-1 "PEH RESP" P1/S7P5P3/
! 49. P3-2 "REPLY WPDNG" R7H4PB/R7C/
! 50. P7C-2 "COMPAT 1 OK" P6/P5E7/
! 51. P7C-3 "COMPAT 1 OK" P6/
! 52. P7-3 "COMPAT 1" P6/
! 53. P6-2 "COMPAT 1" C4/C5/C6/C8/
! 54. C8-1 "EXT LS POS 2" S3/S7S3A/S4R/5S/
! 55. S5-1 "SPLIT PPEP 3" S2/
! 56. S2-2 "SPLIT PPOD POS" C3/
! 57. C3-1 "CH CUE2" ** C2C1S2S1C3S6S5AS4P2S7S3AS3S4AS5SCBC7C6SC4R1/
R3R1OF1R4F2F3PN-5

ADDPPOD (PN-6)
REPPPOD (PN-3)
REPRHSP (PN-2)

REPLY (OK)

SAVED PN-6 PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT NIL

PN-1 "NET"
LMS (LET1 V1) (F V1)
RMS (EXISTS C1 C3) (F1PED (QUOTE PN-1)) (IMAGE (QUOTE DUND)) (LET1 C1) (IN C1)
(LET3 C3) (IN C3) (CUE1P1LE (QUOTE PN-1) C1 (QUOTE X1) C3)
VARS V1
PN-1 "NET"
LMS (LET1 V1) (F V1)
RMS (EXISTS C1) (F1PED (QUOTE PN-1)) (IMAGE (QUOTE DUND)) (LET1 C1) (P C1)
(CUE1P1LE (QUOTE PN-1) C1 (QUOTE X1) (QUOTE X1))
VARS V1
PN-2 "NET"
LMS (LET1 V1) (F V1)

RMS (EXISTS C1 C2 C3) (F1PED (QUOTE PN-2)) (IMAGE (QUOTE POR)) (LET1 C1) (IN C1)
(LET2 C2) (A C2) (LET3 C3) (IN C3) (CUE1P1LE (QUOTE PN-2) C1 C2 C3)
VARS V1
PN-3 "NET"
LMS (LET1 V1) (IN V1) (LET3 V3) (IN V3) (LET2 V2) (A V2)
RMS (F1PED (QUOTE PN-3)) (IMAGE (QUOTE NMI))
(CUE1P1LE (QUOTE PN-3) (QUOTE X1) (QUOTE X1) (QUOTE X1))
VARS V2 V3 V1
PN-4 "NET"
LMS (LET1 V1) (NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (IN V1))
RMS (F1PED (QUOTE PN-4)) (IMAGE (QUOTE NMI))
(CUE1P1LE (QUOTE PN-4) (QUOTE X1) (QUOTE X1) (QUOTE X1))
VARS V1
PN-5 "NET"
LMS (LET1 V1) (IN V1) (LET3 V3) (NOT (IN V3))
RMS (F1PED (QUOTE PN-5)) (IMAGE (QUOTE NMI))
(CUE1P1LE (QUOTE PN-5) (QUOTE X1) (QUOTE X1) (QUOTE X1))
VARS V3 V1
PN-6 "NET"
LMS (LET1 V1) (IN V1) (LET3 V3) (IN V3) (LET2 V2) (NOT (A V2))
RMS (F1PED (QUOTE PN-6)) (IMAGE (QUOTE NMI))
(CUE1P1LE (QUOTE PN-6) (QUOTE X1) (QUOTE X1) (QUOTE X1))
VARS V2 V3 V1

LASTNEW (PN-6)
LMSPEL (PN-1) (LET1 V1)) (PN-1) (LET1 V1)) (PN-2) (LET1 V1))
(PN-3) (LET1 V1) (IN V1) (LET3 V3) (IN V3) (LET2 V2)) (PN-4) (LET1 V1))
(PN-5) (LET1 V1) (IN V1) (LET3 V3))
(PN-6) (LET1 V1) (IN V1) (LET3 V3) (IN V3) (LET2 V2))
LASTERN (PN-1) ((R V1)) POS 1) (PN-1) ((F V1)) POS 1) (PN-2) ((P V1)) POS 1)
(PN-3) ((A V2)) POS 3)
(PN-4) (NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (IN V1)) (NEG 1)
(PN-5) (NOT (IN V3)) (NEG 2) (PN-6) (NOT (A V2)) (NEG 3)
RMSQUE (PN-1) (CUE PN-1 P X1 X1)) (PN-1) (CUE PN-1 N X1 X1))
(PN-2) (CUE PN-2 N A X1)) (PN-3) (CUE PN-3 X1 X1 X1)) (PN-4) (CUE PN-4 X1 X1 X1))
(PN-5) (CUE PN-5 X1 X1 X1)) (PN-6) (CUE PN-6 X1 X1 X1))
RMSIMAGE (PN-1 DUND) (PN-1 DUND) (PN-2 POR) (PN-3 NMI) (PN-4 NMI) (PN-5 NMI)
(PN-6 NMI)

RUN TIME 32.0 SEC

EXAM	TPY	F1PE	WACT	E/T	E/T	T/F
275	B2	57	252	4.02	3.35	1.44
0.116	0.390	0.562	0.127	SEC AVG		

154 INSEPTS 98 DELETES 2 WARNINGS 7 NEW OBJECTS
MAX SHPK LENGTH 34
CORE (FREE.FULL): (8620 . 1836) USED (1322 . 86)

ACTS (LOADPS (EPAM . EXP) (EPAMSR . MAC) EPAMC RUN SHPKEMPTY SHPKEMPTY SHPKEMPTY
SHPKEMPTY SHPKEMPTY SHPKEMPTY SAVES (CLOSED (EPAM1 . EXP)) SAVES (CLOSED (EPAM1 . DBS)) (CLOSED (EPAM1 . TRS)) RUN SHPKEMPTY SHPKEMPTY SHPKEMPTY
SHPKEMPTY SHPKEMPTY SHPKEMPTY

TRACE

(PN-1-1 F2-1 E2-1 E3-1 E1-1 PN-3-1 F3-1 E1-2 R3-1 R7C-1 R7-1 R7-2 R6-1 C7-1
S4-1 S2-1 C2-1 PN-1-1 F2-2 E3-2 E2-2 E1-3 PN-2-1 F3-2 E1-4 E1-5 R1-1 PN-2-2
F2-3 E2-3 E3-3 E1-6 P4-1 E1-7 C2-2 RSA-1 P5-1 E4-1 PN-2-3 F2-4 E2-4 E3-4 E1-8
PN-3-2 F3-3 E1-9 E3-5 R10-1 R3-2 R7C-2 R7C-3 R7-3 R6-2 C8-1 S5-1 S2-2 C3-1)

F1PED 26 OUT OF 48 PRODS

(SAVES EPAM1)
CLOSED (EPAM12 . EXP)

(CLOSED (EPAM12 . DBS))

THIRD SEC OF THREE PAIRS

TOP LEVEL ASSERT (STIM F1R) PN-1-1
! 1. PN-1-1 "NET" F3F2/
! 2. F2-1 "STIM F1PE" P5E2/
! 3. E2-1 "STIM PEH2" E1/
! 4. E1-1 "STIM PEH1" E3/
! 5. E3-1 "STIM PEH3" E4R4F3R3PN-5/PN-3PN-4/PN-1PN-2PN-4/F1P5S5A
SSS4AS4S3AS3

C

Detailed Trace on Three-Pay Test

EPAM

T
TOP LEVEL ASSERT (RESP NUM) SBR255AS457P3R1R571R4/
1 6. R1-1 "REPLY ADS" E4R5E2E1/
1 7. E1-2 "STIM PEM1" E3/
1 8. E3-2 "STIM PEM3" C2/C1/C3/
1 9. E3-1 "CH CUE2" S1S2C1C2C3S6SSAS4P257S3MS3S1AS572R373P5A/
1 10. PSA-1 "PC STIM" R5/
1 11. RS-1 "PE STIM" E4/
1 12. E4-1 "STIM PEM PEM" PN-3PN-5/PN-6/PN-1/
1 13. PN-1-2 "NET" F2/
1 14. F2-2 "STIM FIPE" E2/
1 15. E2-2 "STIM PEM2" R5E1/
1 16. E1-3 "STIM PEM1" E3/
1 17. E3-3 "STIM PEM3" E4R4F3R3PN-6/
1 18. PN-6-1 "NET" F2F3/
1 19. F3-1 "CUE FIPE" E4E3/
1 20. E3-4 "STIM PEM3" E1/
1 21. E1-4 "STIM PEM1" E2/
1 22. E2-3 "STIM PEM2" R5R10/
1 23. R10-1 "PEM RESP" R3/R2/R4S6SSAS457P1/
1 24. R1-1 "REPLY MATCH" R3R2R10R1C272C3C1F3PN-5PN-4

REPRISP (PN-1)

REPLY (OK)

SAVED PN-6 PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT (STIM PAM) PN-4/PN-2PN-1/
1 25. PN-1-1 "NET" F3F2/
1 26. F2-3 "STIM FIPE" R5E3/
1 27. E3-5 "STIM PEM3" E1/
1 28. E1-5 "STIM PEM1" E2/
1 29. E2-4 "STIM PEM2" E4R4F3R3PN-6PN-3PN-1/PN-4/PN-2/
1 30. PN-2-1 "NET" F2F3/
1 31. F3-2 "CUE FIPE" E2/
1 32. E2-5 "STIM PEM2" E1/
1 33. E1-6 "STIM PEM1" E3/
1 34. E1-7 "STIM PEM1" E3/
1 35. E3-6 "STIM PEM3" R5E4R3R2R10R1F3C1C3F2C2PN-5PN-3PN-6F1S3S3MS4S5
R5SSAS4

REPLY (POR)

T
TOP LEVEL ASSERT (RESP POR) F1P5R3/R1/
1 36. R1-2 "REPLY MATCH" R1R10R2R3

REPLY (OK)

SAVED PN-6 PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT (STIM PEM) PN-1PN-2/
1 37. PN-2-2 "NET" F3F2/
1 38. F2-4 "STIM FIPE" R5E2/
1 39. F2-6 "STIM PEM2" E1/
1 40. E1-8 "STIM PEM1" E3/
1 41. E3-7 "STIM PEM3" E4R4F3R3PN-5/PN-3/
1 42. PN-3-1 "NET" F2F3/
1 43. F3-3 "CUE FIPE" E3/
1 44. E3-8 "STIM PEM3" E1/
1 45. E1-9 "STIM PEM1" E2/
1 46. E2-7 "STIM PEM2" R5E4R3R2R10C272C3C1F3PN-6F1S4SS4P5SS4S6S3

REPLY (NUM)

C

T
TOP LEVEL ASSERT (RESP NUM) S7S6R4R3/R5R1/
1 47. R1-3 "REPLY MATCH" R10R1R2R3

REPLY (OK)

SAVED PN-6 PN-5 PN-4 PN-3 PN-2 PN-1 PN-1

NIL

TOP LEVEL ASSERT NIL

PN-1 "NET"
LHS (LETS V1) (F V1)
RHS (EXISTS C1 C2 C3) (FIPE (QUOTE PN-1)) (IMAGE (QUOTE DUNNO)) (LETS C1) (M C1) (LET2 C2) (U C2) (LETS C3) (M C3) (CUE TRIPLE (QUOTE PN-1) C1 C2 C3)
VARS U1
PN-1 "NET"
LHS (LETS V1) (R V1)
RHS (EXISTS C1) (FIPE (QUOTE PN-1)) (IMAGE (QUOTE DUNNO)) (LETS C1) (P C1) (CUE TRIPLE (QUOTE PN-1) C1 (QUOTE XX) (QUOTE XX))
VARS U1
PN-2 "NET"
LHS (LETS V1) (P V1)
RHS (EXISTS C1 C2 C3) (FIPE (QUOTE PN-2)) (IMAGE (QUOTE POR)) (LETS C1) (M C1) (LET2 C2) (A C2) (LETS C3) (M C3) (CUE TRIPLE (QUOTE PN-2) C1 C2 C3)
VARS U1
PN-3 "NET"
LHS (LETS V1) (M V1) (LETS V3) (M V3) (LET2 V2) (A V2)
RHS (FIPE (QUOTE PN-3)) (IMAGE (QUOTE NUM)) (CUE TRIPLE (QUOTE PN-3) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS U2 V3 U1
PN-4 "NET"
LHS (LETS V1) (NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (M V1))
RHS (FIPE (QUOTE PN-4)) (IMAGE (QUOTE NUM)) (CUE TRIPLE (QUOTE PN-4) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS U1
PN-5 "NET"
LHS (LETS V1) (M V1) (LETS V3) (NOT (M V3))
RHS (FIPE (QUOTE PN-5)) (IMAGE (QUOTE NUM)) (CUE TRIPLE (QUOTE PN-5) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS U3 U1
PN-6 "NET"
LHS (LETS V1) (M V1) (LETS V3) (M V3) (LET2 V2) (NOT (A V2))
RHS (FIPE (QUOTE PN-6)) (IMAGE (QUOTE NUM)) (CUE TRIPLE (QUOTE PN-6) (QUOTE XX) (QUOTE XX) (QUOTE XX))
VARS U2 V3 U1

LASTNEW (PN-6)

LHSAPEL (PN-1) ((LETS V1)) (PN-1 ((LETS V1))) (PN-2 ((LETS V1)))
(PN-3 ((LETS V1) (M V1) (LETS V3) (M V3) (LET2 V2))) (PN-4 ((LETS V1)))
(PN-5 ((LETS V1) (M V1) (LETS V3)))
(PN-6 ((LETS V1) (M V1) (LETS V3) (M V3) (LET2 V2)))
LHSTEM (PN-1) ((P V1)) POS 1) (PN-1 ((F V1)) POS 1) (PN-2 ((P V1)) POS 1)
(PN-3 ((A V2)) POS 3)
(PN-4 ((NOT (F V1)) (NOT (R V1)) (NOT (P V1)) (NOT (M V1))) NEG 1)
(PN-5 ((NOT (M V3)) NEG 2) (PN-6 ((NOT (A V2)) NEG 3)
RHSCLUE (PN-1) (CUE PN-1 P XX XX)) (PN-1 (CUE PN-1 M U M)) (PN-2 (CUE PN-2 M A M))
(PN-3 (CUE PN-3 XX XX XX)) (PN-4 (CUE PN-4 XX XX XX))
(PN-5 (CUE PN-5 XX XX XX)) (PN-6 (CUE PN-6 XX XX XX))
RHSIMAGE (PN-1 DUNNO) (PN-1 DUNNO) (PN-2 POR) (PN-3 NUM) (PN-4 NUM) (PN-5 NUM)
(PN-6 NUM)

RUN TIME 22.9 SEC

EXAM	TPY	FIPE	WACT	E/T	E/T	T/T
207	61	47	106	4.40	3.39	1.30
0.111	0.375	0.407	0.123	SEC AVG		

116 INSEPTS 70 DELETES 8 WARNINGS 7 NEW OBJECTS
MAX SMPX LENGTH 70
CODE (FREE/FULL): (BS40 . 1040) USED (1294 . 02)

ACTS LOADS (EPAM . ETP) (EPMSR . PAC) EPWAC RUN SMPXEMPTY SMPXEMPTY SMPXEMPTY
SMPXEMPTY SMPXEMPTY SMPXEMPTY SAVES (CLOSED (EPA1) . ETP) SAVES (CLOSED (EPA1) . DBS) (CLOSED (EPA1) . TFS) RUN SMPXEMPTY SMPXEMPTY SMPXEMPTY
SMPXEMPTY SMPXEMPTY SMPXEMPTY SAVES (CLOSED (EPA12 . ETP) SAVES (CLOSED (EPA12 . DBS) (CLOSED (EPA12 . TFS) RUN SMPXEMPTY SMPXEMPTY SMPXEMPTY

SMXEMPTY SMXEMPTY SMXEMPTY

TRACE

(PN-1-1 F2-1 E2-1 E1-1 E3-1 P4-1 E1-2 E3-2 C3-1 RSA-1 RS-1 E4-1 PN-1-2 F2-2
 E2-2 E1-3 E3-3 PN-6-1 F3-1 E3-4 E1-4 E2-3 R10-1 R1-1 PN-1-1 F2-3 E3-5 E1-5
 E2-4 PN-2-1 F3-2 E2-5 E1-6 E1-7 E3-6 R1-2 PN-2-2 F2-4 E2-6 E1-8 E3-7 PN-3-1
 F3-3 E3-8 E1-9 E2-7 P1-3)

FIRED 17 OUT OF 48 PRODS

(SAVES EPAM)

CLOSED (EPA13 . EXP)

(CLOSED (EPAM13 . DBS))

FOURTH CYCLE THROUGH PAIRS: ALL CORRECT

THREE PAIRS / FOUR CYCLES

STIM FIB / RESP NUM / REPLY OK /

STIM RAM / RESP POP / REPLY OK /

STIM PEK / RESP NAM / REPLY OK /

RUN TIME 44.3 SEC

EXAM	TRY	FIPE	WPACT	E/F	E/T	T/F
376	122	84	372	4.48	3.08	1.45
0.118	0.363	0.528	0.119	SEC AVG		

224 INSERTS 148 DELETES 7 WARNINGS 5 NEW OBJECTS

MAX :SMX LENGTH 33

CORE (FREE.FULL): (8640 . 1813) USED (1302 . 109)

FIRED 28 OUT OF 46 PRODS

STIM FIB / REPLY NAM / RESP NUM / REPLY OK /

STIM RAM / REPLY POP / RESP POP / REPLY OK /

STIM PEK / RESP NAM / REPLY OK /

RUN TIME 32.0 SEC

EXAM	TRY	FIPE	WPACT	E/F	E/T	T/F
275	82	57	252	4.82	3.35	1.44
0.116	0.390	0.562	0.127	SEC AVG		

154 INSERTS 98 DELETES 2 WARNINGS 7 NEW OBJECTS

MAX :SMX LENGTH 34

CORE (FREE.FULL): (8620 . 1836) USED (1322 . 86)

FIRED 26 OUT OF 48 PRODS

STIM FIB / RESP NUM / REPLY OK /

STIM RAM / REPLY POP / RESP POP / REPLY OK /

STIM PEK / REPLY NAM / RESP NAM / REPLY OK /

RUN TIME 22.9 SEC

EXAM	TRY	FIPE	WPACT	E/F	E/T	T/F
207	61	47	186	4.48	3.39	1.30
0.111	0.375	0.487	0.123	SEC AVG		

116 INSERTS 70 DELETES 8 WARNINGS 7 NEW OBJECTS

MAX :SMX LENGTH 29

CORE (FREE.FULL): (8640 . 1840) USED (1294 . 82)

FIRED 17 OUT OF 48 PRODS

STIM FIB / REPLY NUM / RESP NUM / REPLY OK /

STIM RAM / REPLY POP / RESP POP / REPLY OK /

STIM PEK / REPLY NAM / RESP NAM / REPLY OK /

RUN TIME 18.4 SEC

EXAM	TRY	FIPE	WPACT	E/F	E/T	T/F
172	41	34	141	5.06	4.28	1.21
0.107	0.449	0.541	0.130	SEC AVG		

92 INSERTS 49 DELETES 8 WARNINGS 7 NEW OBJECTS

MAX :SMX LENGTH 29

CORE (FREE.FULL): (8615 . 1826) USED (-5562 . 72)

FIRED 11 OUT OF 48 PRODS

(CONTROL FLOW SUMMARY FOR THREE-PAIR TEST)

P F E R C S

F1-1	F	1.
PN-1-1	P	1.
F2-1	F	1.
E1-1	E	3...
PN-1-1	P	1.
F3-1	F	1.
E1-2	E	1.
R10-1	R	2..
PN-1-2	P	1.
F2-2	F	1.
E3-2	E	3...
R4-1	R	1.
C1-1	C	1.
RSA-1	R	2..
E4-1	E	1.
PN-1-3	P	1.
F2-3	F	1.
E3-3	E	3...
PN-1-4	P	1.
F3-2	F	1.
E1-5	E	2..
R10-2	R	4....
C4-1	C	1.
S7-1	S	2..
PS-2	R	1.
E4-2	E	2..
PN-1-5	P	1.
F2-4	F	1.
E2-4	E	3...
PN-2-1	P	1.
F3-3	F	1.
E1-8	E	1.
R10-3	R	6.....
C4-2	C	1.
S3-1	S	2..
PN-2-2	P	1.
F2-5	F	1.
E2-5	E	3...
R4-2	R	1.
C1-2	C	1.
RSA-2	R	2..
E4-4	E	1.
PN-2-3	P	1.
F2-6	F	1.
E1-10	E	3...
PN-3-1	P	1.
F3-4	F	1.
E1-11	E	1.
R10-4	R	6.....
C4-3	C	1.
S3-2	S	2..
PN-1-1	P	1.
F2-1	F	1.
E2-1	E	3...
PN-3-1	P	1.
F3-1	F	1.
E1-2	E	1.
R3-1	R	5.....
C7-1	C	1.
S4-1	S	2..
C2-1	C	1.
PN-1-1	P	1.
F2-2	F	1.
E3-2	E	3...
PN-2-1	P	1.
F3-2	F	1.
E1-4	E	2..
R1-1	R	1.
PN-2-2	P	1.
F2-3	F	1.

E2-3 E 3...
 R4-1 R 1..
 E1-7 E 1..
 C2-2 C 1..
 RSA-1 R 2..
 E4-1 E 1..
 PN-2-3 P 1..
 F2-4 F 1..
 E2-4 E 3...
 PN-3-2 P 1..
 F3-3 F 1..
 E1-9 E 2..
 R10-1 R 6.....
 C8-1 C 1..
 S5-1 S 2..
 C3-1 C 1..
 PN-1-1 P 1..
 F2-1 F 1..
 E2-1 E 3...
 R4-1 R 1..
 E1-2 E 2..
 C3-1 C 1..
 RSA-1 R 2..
 E4-1 E 1..
 PN-1-2 P 1..
 F2-2 F 1..
 E2-2 E 3...
 PN-6-1 P 1..
 F3-1 F 1..
 E3-4 E 3...
 R10-1 R 2..
 PN-1-1 P 1..
 F2-3 F 1..
 E3-5 E 3...
 PN-2-1 P 1..
 F3-2 F 1..
 E2-5 E 4....
 R1-2 R 1..
 PN-2-2 P 1..
 F2-4 F 1..
 E2-6 E 3...
 PN-3-1 P 1..
 F3-3 F 1..
 E3-8 E 3...
 R1-3 R 1..

PERCENTAGES OF FIRINGS OF EACH TYPE, OUT OF TOTAL 100

P 12.....
 F 13.....
 E 37.....
 R 25.....
 C 5.....
 S 5.....

Appendix D. Summary of Behavior for Other Tests

SEVEN PAIRS / THREE CYCLES

STIM PAX / RESP CON / REPLY OK /
 STIM BEK / RESP LUQ / REPLY OK /
 STIM CIT / RESP DER / REPLY OK /
 STIM BLK / REPLY LUQ / RESP MAB / REPLY OK /
 STIM NAL / RESP LEO / REPLY OK /
 STIM REB / REPLY LEO / RESP MOL / REPLY OK /
 STIM NOJ / REPLY LEO / RESP PED / REPLY OK /

RUN TIME 2 MIN. 15.0 SEC

EXAM	TRY	FIRE	IMPACT	E/T	E/T	T/T
1037	316	239	1063	4.34	3.20	1.32
0.130	0.427	0.565	0.127	SEC AVG		

630 INSERTS 433 DELETES 36 WARNINGS 12 NEW OBJECTS

MAX SMPX LENGTH 40

CORE (FREE/FULL): (10366 . 2010) USED (1400 . 32)

STIM PAX / REPLY DUANO / RESP CON / REPLY OK /
 STIM BEK / RESP LUQ / REPLY OK /
 STIM CIT / REPLY DER / RESP DER / REPLY OK /
 STIM BLK / REPLY LUQ / RESP MAB / REPLY OK /
 STIM NAL / REPLY PED / RESP LEO / REPLY OK /
 STIM REB / REPLY MOL / RESP MOL / REPLY OK /
 STIM NOJ / REPLY PED / RESP PED / REPLY OK /

RUN TIME 1 MIN. 49.0 SEC

EXAM	TRY	FIRE	IMPACT	E/T	E/T	T/T
810	248	173	718	4.73	3.30	1.43
0.133	0.439	0.630	0.152	SEC AVG		

420 INSERTS 290 DELETES 10 WARNINGS 15 NEW OBJECTS

MAX SMPX LENGTH 30

CORE (FREE/FULL): (19967 . 2010) USED (1799 . 40)

STIM PAX / REPLY CON / RESP CON / REPLY OK /
 STIM BEK / REPLY LUQ / RESP LUQ / REPLY OK /
 STIM CIT / REPLY DER / RESP DER / REPLY OK /
 STIM BLK / REPLY MAB / RESP MAB / REPLY OK /
 STIM NAL / REPLY LEO / RESP LEO / REPLY OK /
 STIM REB / REPLY MOL / RESP MOL / REPLY OK /
 STIM NOJ / REPLY PED / RESP PED / REPLY OK /

RUN TIME 52.1 SEC

EXAM	TRY	FIRE	IMPACT	E/T	E/T	T/T
396	92	71	304	5.50	4.30	1.30
0.132	0.566	0.734	0.171	SEC AVG		

190 INSERTS 196 DELETES 0 WARNINGS 15 NEW OBJECTS

MAX SMPX LENGTH 37

CORE (FREE/FULL): (19922 . 2005) USED (1844 . 45)

NINE PAIRS / THREE CYCLES

STIM POM / RESP LUB / REPLY OK /
 STIM JUB / RESP YIR / REPLY OK /
 STIM YIR / RESP PIN / REPLY OK /
 STIM YOF / RESP BUD / REPLY OK /
 STIM MUD / RESP MYL / REPLY OK /
 STIM KOV / RESP DUF / REPLY OK /
 STIM UAN / REPLY RIM / RESP MEK / REPLY OK /
 STIM PUF / REPLY LUB / RESP JYB / REPLY OK /
 STIM LUB / RESP KOV / REPLY OK /

RUN TIME 3 MIN. 9.27 SEC

EXAM	TRY	FIRE	IMPACT	E/T	E/T	T/T
1422	430	335	1460	4.24	3.31	1.20
0.133	0.440	0.565	0.130	SEC AVG		

854 INSERTS 646 DELETES 45 WARNINGS 15 NEW OBJECTS
 MAX SMPX LENGTH 42
 CORE (FREE-FULL): (15818 . 1498) USED (4124 . 424)

FIRE 39 OUT OF 56 PRODS

STIM PON / REPLY LUB / RESP LUB / REPLY OK /
 STIM JUB / RESP XYP / REPLY OK /
 STIM VYR / REPLY RIN / RESP RIN / REPLY OK /
 STIM KOF / REPLY BUD / RESP BUD / REPLY OK /
 STIM MUD / REPLY NYL / RESP NYL / REPLY OK /
 STIM KDV / REPLY DUP / RESP DUP / REPLY OK /
 STIM VAN / REPLY MEK / RESP MEK / REPLY OK /
 STIM PUK / REPLY JYB / RESP JYB / REPLY OK /
 STIM LIB / REPLY KOX / RESP KOX / REPLY OK /

RUN TIME 1 MIN. 23.9 SEC

EXAM	TRY	FIPE	WPACT	E/T	E/T	T/T
637	127	90	417	6.5	5.02	1.30
0.132	0.660	0.056	0.201	SEC AVG		

266 INSERTS 151 DELETES 0 WARNINGS 15 NEW OBJECTS
 MAX SMPX LENGTH 37
 CORE (FREE-FULL): (7509 . 1761) USED (2433 . 161)

FIRE 27 OUT OF 56 PRODS

STIM PON / REPLY LUB / RESP LUB / REPLY OK /
 STIM JUB / REPLY XYP / RESP XYP / REPLY OK /
 STIM VYR / REPLY PIN / RESP PIN / REPLY OK /
 STIM KOF / REPLY BUD / RESP BUD / REPLY OK /
 STIM MUD / REPLY NYL / RESP NYL / REPLY OK /
 STIM KDV / REPLY DUP / RESP DUP / REPLY OK /
 STIM VAN / REPLY MEK / RESP MEK / REPLY OK /
 STIM PUK / REPLY JYB / RESP JYB / REPLY OK /
 STIM LIB / REPLY KOX / RESP KOX / REPLY OK /

RUN TIME 1 MIN. 15.7 SEC

EXAM	TRY	FIPE	WPACT	E/T	E/T	T/T
595	113	86	375	6.92	5.27	1.31
0.127	0.670	0.080	0.202	SEC AVG		

244 INSERTS 131 DELETES 0 WARNINGS 15 NEW OBJECTS
 MAX SMPX LENGTH 37
 CORE (FREE-FULL): (7549 . 1770) USED (2393 . 152)

FIRE 21 OUT OF 56 PRODS

 LIST OF SYLLABLES AS SIX PAIRS / THREE CYCLES

STIM PAX / RESP CON / REPLY OK /
 STIM CON / RESP BEK / REPLY OK /
 STIM BEK / RESP LUQ / REPLY OK /
 STIM LUQ / RESP CIT / REPLY OK /
 STIM CIT / RESP LEO / REPLY OK /
 STIM LEO / RESP PAX / REPLY OK /

RUN TIME 1 MIN. 30.5 SEC

EXAM	TRY	FIPE	WPACT	E/T	E/T	T/T
752	232	176	761	4.27	3.24	1.32
0.120	0.390	0.514	0.119	SEC AVG		

451 INSERTS 310 DELETES 15 WARNINGS 7 NEW OBJECTS
 MAX SMPX LENGTH 36
 CORE (FREE-FULL): (7763 . 1707) USED (2179 . 215)

FIRE 33 OUT OF 48 PRODS

STIM PAX / REPLY CIT / RESP CON / REPLY OK /
 STIM CON / RESP BEK / REPLY OK /
 STIM BEK / RESP LUQ / REPLY OK /
 STIM LUQ / RESP CIT / REPLY OK /
 STIM CIT / REPLY LEO / RESP LEO / REPLY OK /
 STIM LEO / REPLY PAX / RESP PAX / REPLY OK /

RUN TIME 1 MIN. 18.1 SEC

EXAM	TRY	FIPE	WPACT	E/T	E/T	T/T
659	174	144	579	4.50	3.79	1.21
0.110	0.449	0.542	0.125	SEC AVG		

346 INSERTS 233 DELETES 4 WARNINGS 9 NEW OBJECTS
 MAX SMPX LENGTH 37
 CORE (FREE-FULL): (7716 . 1734) USED (2226 . 188)

FIRE 31 OUT OF 50 PRODS

STIM PAX / REPLY CON / RESP CON / REPLY OK /
 STIM CON / REPLY BEK / RESP BEK / REPLY OK /
 STIM BEK / REPLY LUQ / RESP LUQ / REPLY OK /
 STIM LUQ / REPLY CIT / RESP CIT / REPLY OK /
 STIM CIT / REPLY LEO / RESP LEO / REPLY OK /
 STIM LEO / REPLY PAX / RESP PAX / REPLY OK /

RUN TIME 45.4 SEC

EXAM	TRY	FIPE	WPACT	E/T	E/T	T/T
366	84	74	300	4.95	4.96	1.19
0.124	0.540	0.613	0.151	SEC AVG		

196 INSERTS 104 DELETES 0 WARNINGS 9 NEW OBJECTS
 MAX SMPX LENGTH 31
 CORE (FREE-FULL): (8186 . 1000) USED (1756 . 114)

FIRE 12 OUT OF 50 PRODS

Appendix E. Matarazzo's EPRI2

1. (READY) (STIM X1) => (REN (READY)) (PERCEIVE X1 ?)
2. (READY) => (ATTEND STIM)
3. (PEPLY) - (PESP) => (ATTEND RESP)
4. (REPLY X1) - (PESP X1) => (PEP REPLY WRONG)
5. (REPLY X1) (PESP X1) => (STOP)
6. (USED) (TEST X1) - (TEST X2) => (REP USED USED=)
7. (TEST X1) (TEST X2) (X3 X4 ?) => (PEN (X3 X4 ?))
8. (TEST X1) (TEST X2) - (R-GEN) => (DEP (REPLY X1) (R-GEN)) (SAY X1)
9. - (RESP) => (DEP (REPLY ?)) (SAY ?) (ATTEND PESP)
10. (RESP X1) - (X2 X3 RESP) => (PERCEIVE X1 PESP)
11. (WRONG) (TEST X1) (STIM X1) - (R-GEN) => (DEP (R-GEN))
12. (OLD X1) (R-GEN) => (PEP OLD COND) (DEP (HOLD X1))
13. (USED X1) (USED=) (R-GEN) => (PEP USED COND) (DEP (HOLD X1))
14. (R-GEN) (COND (X1 X2 ?)) (X1 X2 PESP) => (PEN (X1 X2 RESP))
15. (X1 X2 PESP) (PESP X3) (WRONG X4) - (DONE)
 - => (COND (X1 X2 ?)) (ACTION (OLD) (DEP (REPLY X3)) (SAY X3))
 - (PROD (SAY X4) (TEST X4)) (DEP (DONE))
16. (USED= X1) => (PEP USED= COND)
17. (OLD) (DONE) - (TEST) => (PEP OLD COND)
18. (R-GEN) (HOLD (X1 X2 ?))
 - => (PEN (HOLD (X1 X2 ?))) (ACTION (DEP (X1 X2 ?)))
19. (R-GEN) (X1 X2 PESP) (STIM X3) (WRONG X4)
 - => (ACTION (DEP (TEST X3))) (ACTION (USED) (DEP (X1 X2 ?)))
 - (PROD (DEP (TEST X3))) (STOP)
20. (X1 X2 ?) (X3 X4 RESP) (STIM X5) (WRONG X6)
 - => (COND (X1 X2 ?)) (ACTION (USED) (DEP (X3 X4 ?)) (DEP (TEST X5)))
 - (PROD (SAY X6)) (STOP)

Chapter IV

GPSR

A Production System Implementation of GPS

Abstract. This chapter describes a production system implementation of the General Problem Solver (GPS), a well-known AI program. The new program, called GPSR, consists of over 200 productions and is organized along the lines of the original. The impact of using production systems on representation and control is discussed. Tasks given to GPSR are expressed largely as productions. Consideration of the representation of task knowledge and of the process of encoding the knowledge constituting GPSR's problem-solving executive brings out characteristics of production systems as a language. The behavior of GPSR compares favorably to that of GPS, and it is contended that GPSR would be an appropriate vehicle for further research, with respect to both production systems and problem solving.

Table of Contents

For Chapter IV

SECTION	PAGE
A Introduction	IV-1
Fig. A.1 The Tower of Hanoi puzzle	IV-2
B An Abstract Description of GPSR	IV-5
B.1 An overview of GPSR's components	IV-5
Fig. B.1 Prototype problem solver	IV-5
Fig. B.2 VAPs for the prototype problem solver	IV-5
Fig. B.3 VAPs for the essence of GPSR	IV-7
Fig. B.4 The components of GPSR	IV-8
Fig. B.5 VAPs for GPSR's methods	IV-10
Fig. B.6 The varieties of goal-subgoal structure	IV-11
Fig. B.7 VAPs for canonization processes	IV-11
B.2 A comparison of GPSR and GPS	IV-12
B.3 Production system representations in GPSR	IV-14
C GPSR in Detail	IV-19
C.1 An example of the behavior of GPSR	IV-19
Fig. C.1 Initial trace segment for the Monkey task	IV-20
C.2 The major sets of productions in GPSR	IV-21
Fig. C.2 APs for GPSR executive Ps	IV-22
Fig. C.3 APs for the four kinds of filing	IV-23
Fig. C.4 APs for the method Ps	IV-25
Fig. C.5 APs for the Match-Diff submethod	IV-26
Fig. C.6 APs for low-level processes	IV-27
C.3 Meanings of the GPSR predicates	IV-28
D Tasks Given to GPSR	IV-35
D.1 Justification of choices	IV-35
Fig. D.1 A hierarchy of tasks by method usage	IV-35
D.2 The external representation of tasks for GPSR	IV-36
Fig. D.2 APs for task-specific information	IV-36
D.3 The Monkey task	IV-37
Fig. D.3 Extracts from the RHS of QI of the MK task Ps	IV-37
Fig. D.4 Extracts of operator application and operator difference Ps	IV-38
D.4 The Tower of Hanoi task	IV-40
Fig. D.5 Pre-tests for the MOVE:DISK operator in TH	IV-41
Fig. D.6 Generation of feasible assignments in TH	IV-41
Fig. D.7 Sample Tower of Hanoi situation	IV-42
D.5 The Missionaries and Cannibals task	IV-42
Fig. D.8 Transformed post-tests for MC	IV-43
Fig. D.9 GPSR doesn't need to retry transform goals for MC	IV-45
Fig. D.10 GPSR retries a no-progress goal	IV-46
Fig. D.11 GPSR carries over desired assignments inappropriately	IV-47
D.6 A comparison of task specification in GPS and GPSR	IV-49

Table of Contents

GPSR

E	Production-System-Related Features of GPSR	IV-53
E.1	Low-level implementation features	IV-53
E.2	Trade-offs between Working Memory and Production Memory	IV-56
E.3	The ease of extending GPSR	IV-57
E.4	Mapping the GPS external representation onto the task Ps	IV-59
E.5	The knowledge encoded in the GPSR executive	IV-59
Fig. E.1	Slightly abstract Ps for goal evaluation	IV-60
Fig. E.2	The mapping between N's and E's	IV-61
F	GPS features of GPSR	IV-63
F.1	Features of GPS that can be incorporated into other problem solvers	IV-63
F.2	Problems with GPS from an implementation viewpoint	IV-63
G	Topics for Further Research	IV-67
G.1	GPS research topics	IV-67
G.2	Production system research topics	IV-70
G.3	Production systems as a new level of problem-solving	IV-75
H	References	IV-77
APPENDIX		PAGE
A	Program Listing for GPSR	IV-80
B	Productions for Tasks	IV-87
C	Cross-Reference of Predicates	IV-91
D	Sample Created Net Productions	IV-95
E	Detailed Behavior on the Monkey and Bananas Task	IV-96
F	Behavior Traces for the Other Tasks	IV-101

A. Introduction

This chapter describes a production system (PS) implementation of GPS, the general problem solver of Newell, Shaw, Simon, and Ernst. Of the many versions of GPS reported by various combinations of these four authors, we have chosen the version described in Ernst and Newell (1969). The present version, GPSR (GPS revisited), is implemented in Pslist, for the purpose of exploring the use of PSs as an AI language. However, many issues of interest to pure GPS research have arisen in the course of this project, and they will be treated in the appropriate places in the discussion.

GPS was chosen to test the use of PSs because it represents the broad class of AI programs that solve problems using the heuristic search method. GPS is the most serious effort to achieve generality of approach; the Ernst and Newell (1969) version demonstrated this by solving simple problems in widely separated task domains, including simple symbolic puzzles, resolution theorem-proving, integral calculus problems, parsing strings from a phrase-structured language, and letter series extrapolation. Because of this generality, interesting representational problems are certain to be met. GPS's problem-solving executive, which coordinates the application of problem-solving methods to achieve goals, is probably one of the most complex of the programs in the heuristic search class. Thus, important features of control and representation in PSs will be brought out by this study. Furthermore, GPS has been used as a vehicle for studying human problem solving (Newell and Simon, 1972), so that its implementation as a PS may provide psychological insight and support for PSs as a model of the human control structure. Finally, heuristic search and particular features of GPS have been used recently as a basis for several AI languages (Planner, QA4, Conniver, Popler), so that it will be useful to view GPS as a language or programming system, or at least to consider the ways in which GPS might adapt itself to particular tasks, a more dynamic (automatic programming) process with active participation of the language system in the problem solving process.

GPS went through a lengthy evolution, as described in Ernst and Newell (1969); that work includes a complete bibliography. The version that GPSR mimics was more general than previous versions in its internal representation and in the flexibility of its problem-solving methods, but some advantages were lost as it evolved. For instance, some of the special features for matching and manipulating arithmetic expressions were abandoned (temporarily). GPSR has made only minor attempts to innovate, due to its comparison-oriented purpose, but it is hoped that the version of GPS that it encodes is suitable for further GPS work. Some support for that hope will be given below.

GPSR[•] solves problems in a way significantly better than trial-and-error search: it uses what is called means-ends analysis to focus attention on essential aspects of problems, with the result that its behavior has some semblance of intelligence. A problem is stated as finding a way to transform an initial state to some desired state (or set of states) by manipulations defined by problem operators, given along with the problem

• I will use "GPSR" in the following introductory descriptions, with the understanding that "GPS" could be used just as correctly; any differences between the two will be made explicit.

statement. Means-ends analysis uses a matching process to determine how the present problem state is different from the desired state; it subdivides the task of reducing the differences between the two states into reducing the hardest ones and then working on the easier ones that remain after the hard ones are eliminated. To accomplish this, it must be clear what the means are for obtaining specific ends (reducing specific differences). These means-ends connections are supplied to GPSR, along with hints as to which differences are likely to be hardest to reduce, and it remains for GPSR to carry out the bulk of the manipulations in solving the problem.

An example of a problem for which this approach is suitable is the Tower of Hanoi problem, a puzzle consisting of three fixed pegs and a set of disks of varying sizes that fit on the pegs, where the problem is to move the disks according to a restrictive set of rules from one peg to another. The rules prohibit: moving more than one disk at a time, placing a disk on top of a smaller disk, and moving any but the topmost disk on a peg. This is diagrammed (for the four-disk case) in Figure A.1.

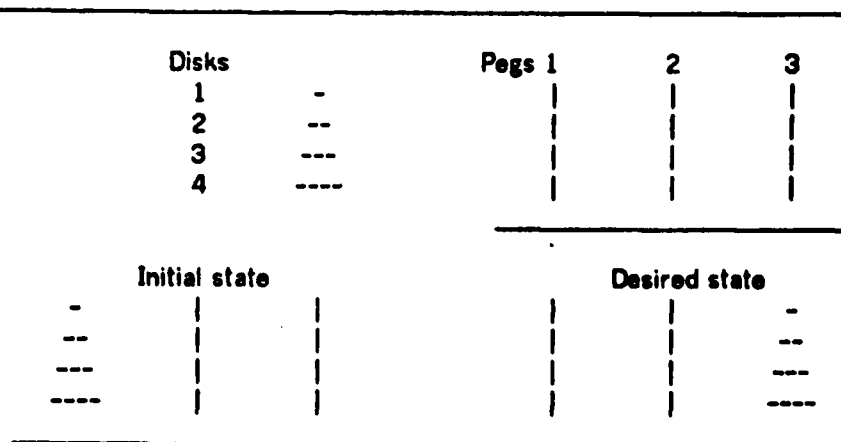


Figure A.1 The Tower of Hanoi puzzle

The statement of the problem for GPSR consists of: a description of the initial and desired state; the statement that the problem is to transform the initial state to the desired state using the MOVE:DISK operator; the description of that operator, including the restrictive rules given above; and the extra information that it is harder to move disk 4 than it is to move disk 3, disk 3 is harder than 2, and 2 harder than 1.

GPSR starts by formulating the problem as a goal to transform the initial state to the desired state. It proceeds by matching the initial state to the desired state, noting that disk 4 is the hardest difference to be reduced, and dividing the initial problem into two subgoals, the first being to move disk 4 from peg 1 to peg 3, the second being to move the rest of the disks. When the first subgoal is attempted, however, it is discovered that disk 4 cannot be moved until disks 1, 2, and 3 are somewhere other than peg 1; so it further subdivides the problem, establishing a subgoal to move disks 1, 2, and 3 to some other peg. We can see that this subgoal is similar to, but easier than, the original, involving fewer disks to be moved. GPSR in fact attacks it in a similar manner, saving the

previous problems it encountered until this sub-problem is solved. It is evident that in stating the problem as described above, a good deal of the work has been left to GPSR; it is this kind of work that is considered to be the primary component of problem-solving in the GPS framework.

This chapter presents GPSR at several levels of detail. Section B gives an overview of the pieces of GPSR and how they fit together. It then contrasts GPSR and GPS at an abstract description level, and surveys how PSs affected the design of GPSR. Section C goes into considerably more detail: it contains an example of the program working, describes the sets of productions (Ps) in the program, and gives meanings for the predicates, the primitive components of the Ps. Section D gives a justification for the tasks chosen as tests for GPSR, and then discusses the behavior of GPSR on the Monkey and Bananas task, on the Tower of Hanoi task, and on the Missionaries and Cannibals task. Section E discusses PS-related features of GPSR, and gives the conclusions on PSs to be drawn from its implementation. Section F gives features of GPS that were elucidated by the experiments. It discusses features of GPS that are useful for other problem-solving programs, and outlines limitations of GPS and difficulties with extant descriptions of GPS. Section G points out topics for further research, from both the GPS and PS standpoints.

B. An Abstract Description of GPSR

B.1. An overview of GPSR's components

Ernst and Newell (1969, page 8) presented the simplified prototype of a problem solver in Figure B.1.

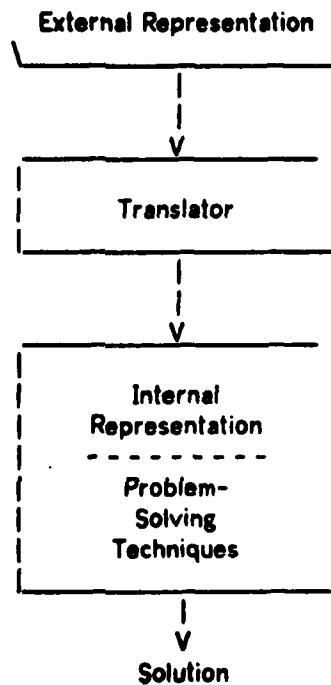


Figure B.1 Prototype problem solver

For our purposes, it is useful to translate this schematic diagram into the PS-like representation in Figure B.2.

PP1: external problem & not represent -> represent & translate & solve;
 PP2: external problem & represent -> external representation of task;
 PP3: translate & external representation of task -> internal representation of task;
 PP4: solve & internal representation of task & applicable techniques -> solution;

Figure B.2 VAPs for the prototype problem solver

There are several points to be noted in this figure. The arrows in PP2, PP3, and PP4 correspond loosely to the three arrows in the block diagram. PP1 represents the assumed step of recognizing the existence of a problem and deciding to proceed according to the stereotyped sequence "represent & translate & solve". PP2's arrow represents the "Representor", PP3's arrow, the "Translator", and PP4's arrow, the "Solver". The control flow represented by the arrows, then, strictly speaking has been mapped to the explicit control signals, "represent", "translate", and "solve". There is assumed in the interpreter of these rules a mechanism for ordering events, so that the "represent" signal is attended to before the "translate", and so on. But it is also the case in this example that the event order is not important (in later rule sets it will be). What is important is that each rule responds to an emerging situation in the presence of a control signal (which in general becomes necessary when global goals establish different modes of response to similar emerging data).

The entities of Figure B.2 will be called very abstract Ps (VAPs). A VAP consists of a condition and an action, but the components of these are abstract, of two types. The first type is close to what it would be in an ordinary P, namely, it is a type for specific signals; this type is not underlined. Entities of the second type represent considerable abstraction and compression, super-conditions and super-actions, as it were; these are underlined. A super-condition may require many condition elements to represent it, it may span many cases, requiring in actuality many Ps to represent it, and it may require many P firings to develop it dynamically. Likewise, a super-action may represent many action elements, and it may require many Ps to effect it. For example, in PP1, "external problem" represents the full collection of external objects and attributes that make up a problem situation; "not represent" means the absence of the "represent" signal; and three signals "represent", "translate", and "solve" are asserted. PP2 expresses the process of representing the external problem, and is a considerable abbreviation.

In the remainder of this section, VAPs are used to represent very concisely a broad outline of the processing done by GPSR. VAPs will indicate the interactions of GPSR's components and their gross behavior rather than giving their internal details. In Section C, abstract Ps (APs) are used to elaborate internal details of the components. Syntactically VAPs and APs are quite similar, but APs are definitely closer to the actual Ps. In APs the number of rules and their structuring into related sets is much closer to the actual Ps than for VAPs. The signals used in APs correspond very closely to actual program predicates, whereas components of VAPs are rather loose expressions of content. The APs generally correspond to VAPs in that super-conditions and super-actions in VAPs are expanded into APs and into component signals. A similar further elaboration occurs between APs and actual Ps.

Before continuing, there are several features of the syntactic and meaning conventions of VAPs and APs that are worth noting. (The reader may want to skim this and refer back to it when necessary.) VAPs have names with two capital letters followed by a number, e.g. PP3, EX1. APs have names that correspond to groups of Ps in GPSR, that is, a capital letter followed by a number, with a lower-case letter suffix. For instance, FOB is the b'th AP in a set corresponding to the O'th group of F Ps, the ones with numbers less than 10. Occasionally, one AP spans a full set, so that it is tagged using "s", e.g. T's; this is also used in the text to refer to a set of APs, e.g. M20's. "Z" is used as a comment character. Arguments to elements are sometimes given, in parentheses, as an unsystematic

clarification to the meaning. In general, only assertion of signals and super-actions is shown; the reader is expected to be able to follow the flow sufficiently well to know that entities are deleted appropriately to avoid looping, needless repetition, and conflicts. The implicit ordering of RHS assertions in the event-based :SMPX of Psnlst will also serve occasionally to control the flow. A common form of element is a selection from a set, which is denoted either by using a superlative adjective in a super-condition, by "arbitrary" in a super-condition, or by "select" in a super-action. An RHS of a VAP may use "OR" to indicate that one of a set of alternative action sequences is done, depending on conditions that are not stated explicitly in the LHS. That is, it is an abbreviation for writing several VAPs, with a further elaboration of the LHS for each of the RHS disjuncts. If an LHS of a VAP contains "OR", it is understood to be an abbreviation for writing each disjunct as a separate LHS with the common RHS. Finally, "s" is used to imply that a set (conjunction) of elements is used in the LHS match or to imply that a set is asserted by the RHS; e.g., "location-link's" for "location-link & location-link & location-link".

All of GPSR is represented by the fourth VAP in Figure B.2. GPS contained a version of the third VAP, but the external representation was in that case already the result of human translation. As illustrations of systems comprising all four VAPs, take the robotics problem solvers, or systems that take natural language input; in both areas present levels of achievement are rudimentary.

% Problem-Solving Executive; corresponds to 24 Ps %
 EX1: focus-on-particular-goal & goal-subgoal-network & goal-status
 -> select-method OR shift-focus-to-another-goal OR propagate-success
 OR propagate-failure;

% Method-Selection; 5 Ps %
 MS1: select-method & goal-attributes -> specific-goal-method;

% Method Execution; 188 Ps %
 MS2: specific-goal-method & goal-attributes & internal-repr-of-task
 -> success OR failure OR new-subgoals-and-status;

where specific-goal-method = {transform-method, reduce-method, moveop-method},
 goal-status = {success, failure, evaluate-new}.

Figure B.3 VAPs for the essence of GPSR

Figure B.3 gives some VAPs for the overall operation of GPSR. The main control in GPSR lies in an executive, whose function is to use present status and past history, stated in terms of goals, to determine what to do to drive the problem-solving process towards a solution. The executive has at its disposal a set of methods for attaining goals, and it uses a tree structure that interrelates goals and their results to decide how to proceed, both in the selection of methods to work on new goals and in deciding how to use the results achieved by the methods. The methods in turn are composed of control decisions and a set of processes used potentially by more than one method; they work with the internal task representation and with information connected to particular goals.

Executive**Inter-relationships of goals (EX1)****Canonization of goals (EX1)****Methods (EX1, MS1)****Internal representation of task (MS2)****Attributes of goals (MS2)****Processes (MS2)****Object comparison and difference (TM2, MD1)****Canonize loc-progs to name differences (TM3)****Operator application and difference (TA1, TA2)****Find locations in objects [QA's, T's]****Apply transformations [QA's, T's]****Operations: for GPSR: ADD:LINK, REM:LINK,****INCR:LINK, DECR:LINK, COPY:LINK;****for GPS: COPY, DECREASE, INCREASE,****MOVE, MOVE-FUNCTION, REMOVE.****Apply tests [QA's]****Relations: for GPSR: encoded directly in Q Ps;****for GPS: FOR-ALL, PARTICULAR, CONSTRAINED-MEMBER,****DEFINED, EQUALS, EXCLUSIVE-MEMBER,****GREATER-THAN, IN-THE-SET, LESS-THAN,****TRUE, and negations of all of the****preceding except the first two.****Canonize loc-progs for operator differences [F's]****Canonize objects resulting from operators [F's]****Desirability and feasibility selection (RM1)****Connect differences and operators [M30's]****Apply loc-prog to TABLE:CONN****Connect variables, differences, and transformations [M30's]****Canonize desired assignments [M30's]****Evaluate differences (TM3)****Apply loc-prog to DIFF:ORDER****Discriminate on the value of the difference [D's]****key: entities in ()'s are VAPs, given in figures in Section B;****in []'s, APs, given in figures in Section C.****Figure B.4 The components of GPSR**

The full structure of GPSR can be described in terms of a set of processes and representations. Figure B.4 shows by outline format the containment and organizational association of these elements. The executive uses three main GPSR elements: information on goals, the goal canonization process, and the methods. The methods use processes that are described in terms of entities from the task environment established by the external representation of the problem. For instance, there are processes to compare objects (representing problem states) and to find differences between them; there are processes that apply the given operators to objects (problem states) to produce new ones; and so

on. Most of the elements of the processes are relatively low-level functions, but are stated in non-GPSR-specific terms. The references to "apply loc-prog" are the only exceptions, because they are implementation-dependent. Details on the processes will be brought out in the discussion in succeeding sections of this chapter.

The portion of GPSR that gives it its distinctive character is neither the executive nor the specific symbolic processes, but its collection of problem-solving methods. GPSR's repertoire of methods is smaller than that of GPS (GPSR can do fewer tasks), but the general flavor is maintained. GPSR has three methods and two submethods: the Transform method, the Reduce method, the Move-Operator method, the Try-Apply submethod, and the Match-Diff submethod. VAPs for these are given in Figure B.5; those VAPs expand MS2 in Figure B.3. The Transform method has as its function to transform an actual object into a desired object. If the two objects are not the same to begin with, it must use the Match-Diff submethod to find differences between the two objects, it must decide which difference is the most difficult, and then it must set up subgoals to reduce that difference in the actual object and to transform the result of that reduction into the desired object. The Reduce method connects the difference to be reduced with a set of operators that might reduce it; it then evokes the Try-Apply submethod to try to apply a member of the set. The Move-Operator method has as its objective to apply a move operator (perhaps only partially specified) to an object. It may be that an operator difference (a difference that blocks the operator's application) has already been determined by the parent goal, in which case a sequence of subgoals is set up, one to reduce that difference and one to apply the operator to the result of the reduction. On the other hand, if no operator difference has been determined, the Try-Apply submethod is evoked to test the operator's applicability. The Try-Apply submethod is given a set of move operators, with variable assignments specified, and an object; it is to determine whether any of the operators can be applied to the object, and if not, to determine the features of the object that block the application. In general, each application failure produces a set of differences; of these, the hardest is saved and the others discarded. Then the result of Try-Apply is to select from these hardest differences the easiest, since that is most likely to succeed, and to carry on with it, sprouting the appropriate subgoals. Finally, the Match-Diff submethod has the function of comparing two structured objects, generating a set of all the differences found at corresponding places in the two objects. How objects are represented will be discussed below, and at that time it will become clearer how Match-Diff works and how its results are expressed.

The kinds of goal relationships that these methods give rise to are presented in Figure B.6. Vertical or diagonal lines are subgoal-supergoal relationships, while horizontal ones indicate the antecedent-goal relationship. The notation "may succeed directly" indicates that the subordinate structures may not be necessary, depending on local favorable conditions.

A variety of data structures are required for GPSR. Objects, which represent complete problem states, are represented as trees, with a top node linked to subordinate nodes by named (task-dependent) relations, and with values at terminal nodes. A path

• GPSR does not have any equivalent of the Form-Operator, Form-Operator-to-Set, Set-Operator, Two-Input-Operator, and Select-Best-Members methods; the remaining GPS methods are incorporated into the executive, as discussed in the next subsection.

% Transform method; 14 actual Ps %

TM1: transform-method & objects-have-same-name -> succeed;

TM2: transform-method & not objects-have-same-name -> match-diff & check-match-result;

TM3: check-match-result & hardest-diffr-between-objects
-> create-subgoal-to-reduce-diffr & transform-that-result;

% Reduce method; 20 Ps %

RM1: reduce-method & diffr-to-be-reduced & opr-to-reduce-diffr's
& internal-repr-of-opr's

-> try-apply-opr & opr-with-desired-and-feasible-assignments's;

% Move-operator method; 6 Ps %

MO1: moveop-method & opr-diffr-given

-> create-subgoal-to-reduce-diffr & apply-opr-to-that-result;

MO2: moveop-method & not opr-diffr-given -> try-apply-opr;

% Try-Apply submethod; 19 Ps %

TA1: try-apply-opr & opr-immediate-applicable -> succeed;

TA2: try-apply-opr & submethod-of-reduce-method & easiest-hardest-opr-diffr
-> create-subgoal-to-apply-opr-given-that-opr-diffr;

TA3: try-apply-opr & submethod-of-moveop-method & easiest-hardest-opr-diffr
-> create-subgoal-to-reduce-that-opr-diffr & apply-opr-to-that-result;

% Match-Diff submethod; 11 Ps %

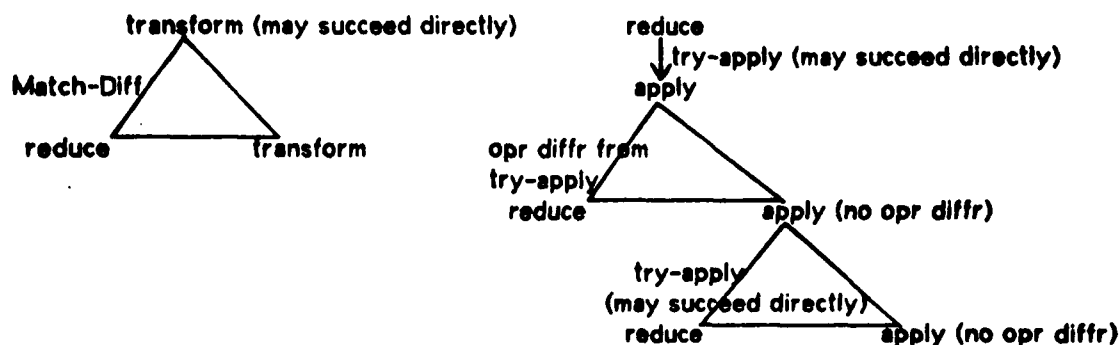
MD1: match-diff & objects-to-be-matched -> diffr-between-objects's;

diffr = difference; opr = operator; repr = representation;
inputs from executive: transform-method, reduce-method, moveop-method;
outputs to executive: succeed (fail not shown), transform-that-result,
apply-opr-to-that-result;

Figure B.5 VAPs for GPSR's methods

from the top node to a terminal is given by a list of the links along the path; this link path is called a loc-prog. Differences are expressed by giving a loc-prog plus two contrasting values, one of which is the actual value in an object and the second of which is the desired value, or the corresponding value in the matched object. For instance, in the Tower of Hanoi object, the hardest difference detected in matching the initial object and the desired object (Figure A.1) is that at location (PEG3 DISK4) there is nothing, UNDEF, where the desired value is YES; in other words, the largest disk is not on the third peg which is its desired location.

Goals in GPSR are names with sets of associated properties. All goals have: an actual object; a supergoal, from which the goal was derived; a marker giving the goal type (transform, reduce, or apply); and a numerical difficulty value (defined later). In addition, a transform goal has a desired object, and it may have an antecedent goal (see Figure B.6). A reduce goal has a difference to be reduced, in addition to the common properties. An apply goal has an operator to be applied and a desired assignment for that operator's



opr diff = operator difference.

Figure B.6 The varieties of goal-subgoal structure

variables; it may have an antecedent goal, and it may be given an operator difference to be reduced. A desired assignment is a partial or full assignment, a set of simple associations of values with move-operator variables.

GPSR canonizes three types of data structures: objects, loc-progs, and desired assignments. The canonization process is necessary to facilitate the recognition of repeated goals; a goal with the same attributes as some previous goal (except supergoal and antecedent-goal properties) is abandoned to avoid loops in the search. Goals themselves are not canonized because of their uniform task-independent representation, but are recognized, if repeated, by a special-purpose set of Ps. *The three entities that are canonized are dissimilar, but the Ps to achieve the canonization can all be described by the VAPs in Figure B.7.*

% Canonization; loc-progs, 6 Ps, objects, 34 Ps, desired assignments, 9 Ps %
 CA1: file-entity -> test-entity-net & check-net-test-result;
 CA2: check-net-test-result & entity-present -> old-entity;
 CA3: check-net-test-result & not entity-present -> add-new-entity-to-net;

% entity net %
 EN1: test-entity-net & entity-matching -> entity-present;

Figure B.7 VAPs for canonization processes

The entities are stored in a recognition network inspired by EPAM (Feigenbaum, 1963). That is, a network is constructed to recognize occurrences of known objects. The recognition may be exact, or partial, based on a proper subset of features of an object; in the latter case, further matching must be done on the candidates proposed by the network.

We have now discussed the top four levels in the outline of GPSR components in

Figure B.4, tying them together by specifying an abstract model of GPSR. The detail of the lower levels will be discussed in Section C. The main differences between GPSR and GPS can be dealt with at the abstract model level, which we now proceed to do.

B.2. A comparison of GPSR and GPS

The major difference between the methods of GPSR and those of GPS is in the way the Try-Apply submethod works. GPSR tries all of the ways of applying an operator, as given by the possible feasible assignments, and then selects the easiest way to proceed; that is, it finds all of the possible operator differences before proceeding. The exception to this is in case an operator applies to an object, with no difference produced, in which case the goal that Try-Apply is working on succeeds. In the analogous situation, GPS tried to apply an operator until an operator difference of tolerable difficulty was found, and proceeded to work on that without searching for other, possibly easier, differences (see Ernst and Newell, 1969, p. 111). In principle, the GPS approach is more conservative, since there is no guarantee that an undesirable number of assignments might not have to be tried, to find all of the operator differences. In practice, this limitation is inoperative; the largest number of feasible assignments generated by GPSR tasks at one time was five. The reason for this difference is probably the accidental historical development of GPS. The construction of GPSR took advantage of the hindsight provided by the implementors of GPS; the modification seemed to be reasonable and an interesting one to try. Furthermore, as it is currently implemented, it would be easy to convert GPSR to the more conservative strategy. In fact, there seems to be little difference attributable to this decision, at the external behavior level (the tasks given to GPS and GPSR do not exercise this capability - there is no room for serious mistakes or gains within those simple tasks).

Several differences between GPS and GPSR deal with the problem-solving executive. The executive in GPSR incorporates four bodies of expertise that were described as methods in GPS: the Try-Old-Goals method, the Antecedent-Goal method, the Expanded-Transform method, and the Transform-Set method. These all deal with selecting goals for further effort, and thus are intimately connected with the executive. In fact, all but the Antecedent-Goal method are separable, comprising a total of about six Ps; there is no reason why they should be tied into the method-selection process instead of being evoked directly by other executive Ps. The Antecedent-Goal method deals with the selection of an antecedent of a goal in case of failure; this is so closely related to selecting the super-goal of a goal under the same circumstances (a function of the executive in GPS) that the decision to merge it into the executive is justified. The only alternative would be to remove all goal-selecting expertise from the executive, collecting it into methods; this in fact is the logical extension of the principles that GPS seems to embody in part in the construction of its methods.

GPSR has nothing corresponding to the method language of GPS. That language was interpreted by the executive; the methods expressed in the language communicated with the executive by means of specific signals; and in some cases the executive could nest invocations of various method-language routines in a recursive way, suspending one to descend into another, and then returning to the original routine, to continue to its next step. There is one place in GPSR where that form of sequencing of steps of a method is found to be necessary: in sequencing the creation of antecedent-succedent goal pairs, for

instance a reduce goal followed by an apply goal using the result of the reduce. In GPSR, this was simply implemented as a data signal indicating the course of action on the success of the antecedent goal. If the antecedent failed, that signal would not be used, and the executive would have no need to pop itself out of a level of method code. (I am imposing an interpretation on how GPS's executive worked that may not correspond to actual implementation details; it is based on figure 12 in Ernst and Newell, pp. 44-45.) Regardless of the mechanics of its interpretation, the method language did give GPS a useful descriptive character. GPSR's methods are expressed directly as Ps; the level at which the programming is done is similar to that of GPS's method language, although at many points the Ps must be concerned with more of the details of the data structures involved. In fact, the VAPs used above to describe GPSR's methods correspond more favorably to the method language. We have noted above the numbers of Ps corresponding to the VAPs given; we will see below how the VAPs and Ps actually correspond (Section C).

GPSR assigns difficulties to goals differently from GPS, and its propagation of difficulties to related goals is slightly different. Difficulties are used to ensure that progress is maintained: proceeding from a goal to one that is more difficult is not allowed (although the rejected goal may be selected later as part of the Try-Old-Goals method, but then only if no easier goals are available). Difficulties originate from goals that have differences associated with them, and are propagated to some of the goals adjoining them in the goal-subgoal network. The numerical scheme used in GPSR to evaluate differences (see the end of Section C.2) is not the same as in GPS, based on behavioral differences: GPSR sometimes rejects goals as too difficult (see Section D.5) where GPS doesn't. Ernst and Newell do not describe their exact difference-evaluation formula. With respect to propagation of difficulties from goals with differences, GPSR and GPS propagate them from subgoals to supergoals, but GPSR also propagates from antecedent goals to their consequents. This difference combines with the difference in difficulty evaluation to produce behavioral differences.

Objects are represented differently in GPSR. As described above, objects are simply trees; the links between nodes in the trees are totally task-dependent; they are also uniform. GPS, on the other hand, distinguished between known links, used by the system over all tasks, and task-dependent ones. It also referred to some subtrees as local description lists. These distinctions no doubt arose from the internal representations used: Psnlst is limited to, at best, imitating a description list by a set of associations; the uniform tree representation chosen is a good way to do this. The known links used by GPS were "first", "second", etc. These were strictly ordered for GPS, whereas GPSR's trees have no such order. Order is not necessary in GPSR, since all objects for tasks given to GPSR are fixed in format, so that corresponding links are always found in two objects, and there are no alternatives. The transition to more general objects would require more care in the matching process to place the right branches of trees in correspondence (to be discussed in more detail in Section G).

The canonization of the various data structures is more varied and specialized in GPSR than in GPS. It is impractical to use a single type of data structure to represent the various entities that need to be recognized (goals, objects, desired assignments, and loc-progs) because Psnlst's single-level representation and limited match indirection prohibit the effective use of a representation with the requisite structure and generality. The

representation doesn't allow nested structure, but instead gets structure indirectly by naming a node in a structure and attaching other single-level attributes to it, including further structural pointers. A Pslist version of an arbitrary description list can be made up, but then the Ps that use such structures would have to pay a cost for the indirection in the representation. Instead, each data structure type is specialized, with structural links and conventions to suit the exact traits of the structure (see the following subsection and Section C). Thus there are three kinds of recognition nets built by GPSR: one for loc-progs, one for objects, and one for desired assignments. Goals must also be recognized, but these are even more of a special case: since their structures are fixed, pre-coded Ps can be used to compare a new goal to others of the same type (transform, reduce, or apply). The other three data structures for which canonization is necessary are task-dependent with variable structure, so that pre-coded Ps are unsuitable.

The final major difference between GPSR and GPS is in the external representation of tasks. Without going into detail here (see Section D), we can say that the external language used by GPS has been bypassed: operators and other active elements of a task representation are encoded directly as Ps, and other information pertaining to the problem to be solved and its structures and operators is expressed directly as Working Memory items. There are two reasons for this: the representation chosen for GPSR is approximately of the same conciseness (in terms of space on the page) as for GPS, due to Ps' inherent high-level character; GPS's language itself was rather artificial and not close to what would be a natural language expression of the tasks. So, rather than deal in two artificial languages of approximately similar level, the direct P notation was adopted. We will discuss further below (Section D.6, Section E.4, and Section G) the interesting topics of how the external problem gets mapped onto the task Ps, and how this relates to natural language translation.

B.3. Production system representations in GPSR

The use of PSs has an impact on a variety of representational issues in GPSR. This subsection emphasizes and brings together the aspects of representation that are necessarily fragmented elsewhere due to different organization. At the same time the presentation here is abstract, so that if the reader needs more detail, he must piece it together after reading later sections. The principle that has guided the use of PSs is not so much the application of PSs in a uniform way, but rather the use of the language facilities in an expedient, concise, and efficient way. First, we consider networks of decisions, which includes method selection, canonization, connecting differences with operators, and evaluating differences. Second, we look at the representation of the operation of accessing something in an object. Third, we see that PSs provide an interesting approach to implementing the GPS match. Fourth, we consider goal and control contexts. Finally, we discuss selection and generation.

Networks of decisions (discrimination nets) appear in several places in GPSR, and their implementation varies to suit particular processing demands. The method selection process, which is a fixed part of GPSR, is encoded as a set of Ps, the result of each of which is a signal to evoke a particular method. As was sketched in Figure B.3, these Ps

• cf. Ernst and Newell, 1969, p. 47

use whatever goal attributes are necessary to make the selection unique. Each conjunct in the condition of a P represents, as it were, an additional node in a tree; Ps whose conditions have a nontrivial intersection can be thought of as sharing a common path for some tests and then branching to their respective terminals.

The canonization of objects, desired assignments, and loc-progs is achieved by constructing Ps whose conditions include the specific constants that compose the entities to be recognized. These Ps are constructed by GPSR as the solution process proceeds, because they contain task-dependent information (constants tested by conditions) and because their size (number of conditions) is dependent on corresponding (variable) properties of task entities. The canonization of goals, on the other hand, does not need to be by program-constructed Ps because goals have fixed, task-independent properties and because data on goals is kept in Working Memory. A goal-recognition P compares a new goal to all other goals, progressively narrowing down the set of possibly identical goals as each conjunct of the condition applies its restriction. Thus a goal-recognition P, by using variables where the other program-constructed nets use constants, represents a set of Ps, and can be thought of as a net schema, that is, as a prototype or template for a set of Ps composing a net.

The representation of the table of connections between differences and the operators that might reduce them, and of the ordering of differences according to expected difficulty of their reduction, are similar in GPSR. They are represented as objects, TABLE:CONN and DIFF:ORDER, and their content is extracted by applying loc-progs to them. This representation is schematic in the same sense as the goal-recognition Ps are: each object could equivalently be expressed as a set of Ps, with LHSs that test attributes of differences. Notationally, the object (schematic) representation is more easily specified in the external representation of tasks, although it would be straightforward to convert such objects to sets of Ps. Convenience has dictated maintaining the present status until task contingencies force generalization to TABLE:CONN's and DIFF:ORDER's that are not so concisely expressable. It is also apparent from this that objects representing problem states could also be generalized to sets of Ps whose properties would become evident through evocation or activation rather than by being passively examined.

The reason why the present solution for the TABLE:CONN and DIFF:ORDER representation is acceptable is that the application of a loc-prog to an object is achieved with a single P. The use of single Ps to access and test values in objects is common to loc-prog application, operator application, and operator difference. When a loc-prog is canonized, a P is constructed whose firing will apply the loc-prog to an object to get a terminal value. This P includes the constants and conditions necessary to follow the link-path from the top node of the object to a terminal node. Operator application and operator difference (specified externally for each task) are also single Ps that include the necessary conjuncts to find terminal values within objects; this capability is used to apply tests from operator pre-conditions, operator post-conditions, and operation-applicability conditions.

The match in GPSR (Match-Diff submethod) is not able to take advantage of the powerful PS match because the PS match is not able (at the present stage of development) to extract differences. Instead the approach taken is similar to a recursive approach: for a node, match up all links to subordinate nodes, and then apply the match to those. There

are a variety of conditions to be tested that are expressed as separate Ps. The Penlist implementation of this match doesn't have the rigidity of a LISP recursive match, in that things are not tried in any particular order. Another important factor is that of Penlist's match finding all possible matches to a P; this means that if several nodes in an object have similar characteristics, the match of the appropriate P carries forward the match on all of them simultaneously. The match has the property that its results (differences) are not necessarily returned at the same time; this is used to advantage by a variant of the match that seeks only one difference: it takes the first one produced and erases all the remaining match signals so that no further work is done. One other feature of the match can be mentioned: matches to described objects are evoked in the same way as matches to other objects, but the representation of described objects changes the process. Described objects are represented as Ps that perform the tests that constitute their descriptions, responding to the signals that initiate the Match-Diff submethod.

The PS approach to goal contexts seems to differ from that used in GPS. In GPSR, the contexts are always present, as Working Memory items containing the name of the goal to which they relate. Thus, context-switching is invisible, whereas in GPS (judging from its prominence in the description) some degree of effort was involved. It is invisible in GPSR, no doubt, because the context is established automatically on doing a P match. The program control context is also treated somewhat differently. There is no method language and no interpreter to maintain control stacks for nested method evocations; rather, when a method logically branches into two submethods, the first is evoked directly and a data signal is left to indicate what is to be done on success of that first method; the data signal is used by the executive. Thus there is no control hierarchy as such, but data signals are used to recommend sequences of action to the executive.

The PS approach is used to advantage in easily specifying complex selection processes and combinatorial data generation. There are no select goals in GPSR: the selections that exist are done by small groups of Ps. Usually one P is sufficient to do the bulk of the selection; this is the case for the Try-Old-Goals process, which selects a goal for further problem-solving effort, and for the selection of new objects for creation of Transform goals (in GPS, the Transform-Set method). More than a single P is necessary in the case of a cascade of selections, for instance the selection in the Try-Apply submethod that first collects the hardest operator difference from a particular feasible assignment, and then for all feasible assignments selects the easiest. The use of a cascade of Ps to make a selection is partially a question of convenience, since it is possible to pile up in a single P the conditions for the logical combinations that constitute the selection. The generation of combinatorial possibilities by multiple outputs from a single match is used in generating feasible operator assignments. A single P condition has conjuncts that specify elements from several sets (domains of variables), and the result of the match is to combine those elements in all of the possible ways. Generally, this feasible assignment generation also includes some conditions to reject certain of the combinations before they're emitted.

In summary, there are several useful attributes of PSs that have been brought out. In the case of the recognition and selection networks, Ps provide a near-ideal form, and it is possible for a PS to build such networks in a task-dependent fashion. Also, PSs are capable of varying degrees of expression schematically. The use of single Ps to access and test values in objects is an indication of the power of the PS match. The GPSR match

(Match-Diff) implementation illustrates control flexibility and openness. The power of the match and of the global Working Memory with respect to establishing local contexts and control sequences is evident from examining goal contexts and method sequencing. And the power of the match allows complex selections and generations of data to be done easily.

C. GPSR in Detail

C.1. An example of the behavior of GPSR

This subsection will go through in detail an example of the behavior of GPSR solving the monkey and bananas problem. This example was chosen because it is relatively brief, yet exhibits many of the important features of GPSR. Appendix E exhibits the full trace printed by GPSR in solving the problem, the final state of the Working Memory after it finished, the complete trace of the Ps that fired, and a control flow summary diagram of that P trace. Figure C.1 gives the segment to be discussed here. The first segment of Appendix B has the task-specific Ps. In the following, we will be referring almost exclusively to the trace printed by GPSR. Entities in []'s refer to the associated VAPs, Figure B.3 and Figure B.5. Note that this discussion leaves out reference to much detail, including the task-specific Ps.

The initial object (situation) in this task consists of a monkey at a certain place, PLACE1, a box at another place, PLACE2, and some bananas at a third place, above a place denoted UNDER:BANANAS. This is represented by (MONKEY:PLACE PLACE1 BOX:PLACE PLACE2). The only other attribute of problem objects for this task is the MONKEY:HAND attribute, which is left undefined in the initial object; it is placed in an object during the problem solving process whenever an operator puts something in the monkey's hand. Other aspects of the problem are encoded directly in the operators and in the desired situation, including the existence and location of the bananas, oddly enough, following the original Ernst and Newell formulation. The desired situation is a described object that specifies that the monkey has the bananas in its hand. GPSR must transform the initial object into the desired one by applying in an appropriate sequence the following operators: CLIMB, which requires that the monkey and the box be at the same place, and which results in the monkey's being on top of the box; GET:BANANAS, which requires that the monkey be on the box and under the bananas, and which results in the monkey's having the bananas in its hand; MOVE:BOX, which requires that the monkey and box be at the same place, and which results in changes in location of both box and monkey, according to the value of the variable MOVE:TO; and WALK, which simply changes the location of the monkey to the value assigned to the variable WALK:TO. GPSR has two versions of the monkey task table of connections (TABLE:CONN); the one we'll discuss here specifies that whatever difference is being reduced, all four operators are equally desirable. (The other version restricts the choice of operators.) The differences that can be encountered are ordered (DIFF:ORDER) as follows: the hardest to reduce is the contents of the monkey's hand; next hardest is the box's location; and easiest is the monkey's location.

GPSR starts out by doing some initialization. It prints out in the trace what its top goal (G-1) is (see Figure C.1), files (canonizes) the initial object, evaluates G-1, and decides to proceed based on a favorable difficulty comparison (initially the top goal has a difficulty level of 0) [see VAP EX1]. The next step is to select a method to work on G-1 [MS1]. The method-selection chooses the Transform method, according to G-1's type. The Transform method proceeds [TM2] by matching [MD1] the initial object to the desired object, finding that the monkey's hand does not have anything. (In the process of finding this difference,

```

G-1 : TRANSFORM INITIAL OBJECT TO DESIRED OBJECT (FROM TOP)
LOC:PROG LP-1 (MONKEY:HAND)
  G-2 : REDUCE UNDEF TO BANANAS AT (MONKEY:HAND) OF INITIAL OBJECT (DIFFIC 305) (FROM G-1)
  LOC:PROG LP-2 (MONKEY:PLACE)
  LOC:PROG LP-3 (BOX:PLACE)
  G-3 : APPLY CLIMB TO INITIAL OBJECT (DIFFIC 100) (FROM G-2)
  ASSIGNS DUMMY ← BANANAS
  G-4 : REDUCE PLACE1 TO PLACE2 AT (MONKEY:PLACE) OF INITIAL OBJECT (DIFFIC 100) (FROM G-3)
  APPLY WALK TO INITIAL OBJECT GET O-1 (WALK TO PLACE2)
  G-4 SUCCEEDS
  G-5 : APPLY CLIMB TO O-1 (DIFFIC 100) (FROM G-3 AND G-4)
  ASSIGNS DUMMY ← BANANAS
  O-1 (BOX:PLACE PLACE2 MONKEY:PLACE PLACE2)
  APPLY CLIMB TO O-1 GET O-2
  G-5 SUCCEEDS
  G-3 SUCCEEDS
  G-2 SUCCEEDS
  G-6 : TRANSFORM O-2 TO DESIRED OBJECT (FROM G-1 AND G-2)
  O-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON BOX)

```

Figure C.1 Initial trace segment for the Monkey task

a new loc-prog, for Monkey:Hand, is discovered and filed, as indicated in the second line of the trace.) The difference gives rise [TM3] to G-2, a goal to reduce it. After the Reduce method is selected [EX1, MS1] the table of connections gives GPSR the set of four operators; from the given information on the operators, it constructs desirable assignments for the operators' variables (CLIMB and GET: BANANAS have only dummy variables) and proceeds, using the Try-Apply submethod to apply them [RM1]. In the process of constructing desirable assignments, enough information about the operators WALK and MOVE:BOX is given to allow GPSR to reject considering them further (GPSR knows that they can effect changes in location only); but for CLIMB and GET: BANANAS, there are only dummy variables, so that it can know nothing about their effects without trying them (In other words, that is the extent of action of the desirability selection process; this is deficient, in ways to be discussed in Section D.3).

Continuing with the attack on goal G-3, Try-Apply finds operator differences [TA2] for CLIMB (the monkey's place isn't the box's place) and GET: BANANAS (the box is not under the bananas); the latter is a more difficult difference, as given in the DIFF:ORDER object, so G-3, a goal to apply CLIMB, is set up (since both operators seem equally likely to get a desired result, the one that looks easier to apply is chosen). Try-Apply has already determined the operator difference, so that G-4 is immediately created [EX1, MS1, MO1] to reduce the difference in the monkey's location, from PLACE1 to PLACE2. This time the desirability selection [EX1, MS1, RM1] has more information to go on, and is able to specify desirable assignments for WALK and MOVE:BOX. Try-Apply finds the latter infeasible, but the former is applicable without operator differences [TA1] and G-4 succeeds with the monkey walking from PLACE1 to PLACE2; the new object (situation) is called O-1 and is filed in the net for objects (it is listed in the trace after G-5).

The executive [EX1], on success of G-4, finds the signal left by G-3: as an apply goal, it was reduced to a sequence of reducing an operator difference and then applying

the operator. Thus the executive creates G-5 from the result of G-4; G-5 is goal to apply CLIMB to O-1. The operator difference for CLIMB has genuinely disappeared, and there were no unforeseen side-effects of the WALK operator, so that CLIMB is applicable [EX1, MS2, MO2, TA1], producing O-2 (listed in the trace after G-6). O-2 is filed, and G-5 succeeds; the executive [EX1] propagates the success back up to G-2, the reduce goal sprouted in the attempt to attain G-1. G-2 is the first goal in a two-element sequence: it is followed by a transform goal on its result, and achieving that transform goal amounts to achieving G-1. So we have G-6, to transform O-2 to the desired object.

The foregoing has provided enough detail to tie together the structure of GPSR as presented in the VAPs in Section B. The reader should now be able to follow the GPSR behavior traces. For the ambitious reader, the foregoing also provides details that should prove useful in following the system in full detail; the remainder of this section describes the components of GPSR whose workings are essential to that endeavor. The Monkey task is discussed further in Section D.

C.2. The major sets of productions in GPSR

This subsection will present the Ps of GPSR in nine sets, which include everything except the task-specific Ps (which are the Q's, see Section D). These sets correspond to labelling conventions, according to initial letters of P names: E's for the executive Ps, F's for filing (canonization), M's for methods, K's for matching (comparing), T's for applying transformations, C's for copying objects, D's for evaluating differences, V's for tracing (viewing) the program's operation, and X's for building external representations of objects. For each P group a set of abstract Ps (APs) elaborates on the VAP structure. For more detail, the Ps themselves (Appendix A) must be consulted; in addition, Section C.3 gives meanings for the predicates used in the Ps.

The executive Ps are divided into four groups, indicated in Figure C.2. The eleven APs for the executive correspond to the VAP EX1 (Figure B.3). They also correspond to 24 actual Ps. The E0 APs represent the evaluate-goal process: action is taken according to whether a goal is too difficult, is a repetition of a previous goal, or is neither. E0a is the initialization of GPSR, evoked at the beginning of a task. The E10 APs take action to propagate the success of a goal, either setting up the second goal of a goal sequence (e.g. reduce-transform) or causing the supergoal to succeed. The E20's propagate failure by retrying a goal from which the failed goal was derived, or by evoking a try-old-goals selection. The E30's are the try-old-goals selection. This includes selection by the New-Obj criterion, which sets up a goal to transform some selected object into the desired object; only objects for which such transform goals do not already exist are candidates. The executive is initially evoked by an "eval-goal" signal from a task-specific initialization P. It passes control in two ways: to the goal-filing process, using "file-goal" (see APs E10a, E10b, and E30b), and to the method-selection process (E0b, E20a, and E30a). Control is passed to it by "eval-goal", "succeed", and "fail" from methods, and by "repeated" (when appropriate) from the goal-filing process. The New-Obj criterion is turned on externally by the user.

The filing Ps are divided into four groups: a group to file loc-progs, a group to recognize goals, a group to file objects, and a group to file desired assignments. The

% E0's: goal evaluation and initialization; 7 Ps %
 E0a: gpsr-init -> initialize;
 E0b: eval-goal & diffic-ok & not repeated -> select-method;
 E0c: eval-goal & diffic-too-high & not repeated
 -> abandon & retry-supergoal-or-antecedent;
 E0d: eval-goal & repeated -> try-old-goals;

 % E10's: proceed from success; 3 Ps %
 E10a: succeed & next-trans -> file-goal & eval-goal & create-new-trans;
 E10b: succeed & next-apply -> file-goal & eval-goal & create-new-apply;
 E10c: succeed otherwise -> succeed(supergoal);

 % E20's: fail; 8 Ps %
 E20a: fail & (antecedent retryable OR supergoal retryable) -> select-method & retry;
 E20b: fail otherwise -> fail OR try-old-goals;

 % E30's: Try-Old-Goals process; 6 Ps %
 E30a: try-old-goals & not new-obj-criterion & arbitrary-reduce-with-least-difficulty
 -> select-method & retry;
 E30b: try-old-goals & new-obj-criterion & oldest-object-not-in-trans
 -> file-goal & eval-goal & create-new-trans;

where trans = transform goal, reduce = reduce goal, apply = apply goal.

Figure C.2 APs for GPSR executive Ps

three filing groups (goal recognition excluded) are expansions or images of the VAPs CA1, CA2, CA3, and EN1 of Figure B.7. Seventeen APs (Figure C.3) correspond to sixty Ps, not including the five types of Ps constructed by GPSR that are represented by F0b, F0c, F0d, F10b, and F50b. Entry to the filing processes is gained via the AP signals "file-loc-prog", "file-goal", "file-object", and "file-des-asg". The file-loc-prog process returns by emitting the name of the loc-prog; the file-goal process emits a "repeated" signal if appropriate; and the file-object and file-des-asg processes return by emitting updated versions of data instances containing the recognized entity. For other kinds of return (i.e., when no definite recognition took place) control falls back (passively) to the evoking process when the filing process runs out of things to do (more precisely, the Psnlst stack :SMPX no longer has events relevant to filing).

The loc-prog filing process builds three Ps for each loc-prog: one to recognize it and name it (F0b), one to apply it to an object (F0c), and one to emit its components given its name (F0d). In each case the P includes full information; this is not like an EPAM discrimination net which decides on the basis of a subset of distinguishing characteristics. Loc-prog filing (F0a, F0e) and recognition (F0b) are done to name the differences resulting from Match-Diff (K's) and to name operator differences detected by task-specific operator Ps (QD's). F0c is evoked by the Reduce method to access TABLE:CONN (M30a) and by difference evaluation, to use information in DIFF:ORDER (D0a). F0d is used by the desirability selection process in the Reduce method, M30c.

• Some Ps constructed by GPSR are given in Appendix D.

- 7 F0's: file loc-progs; 6 Ps 7
- F0a: file-loc-prog & location-link's -> location-link's & extend-net;
- 7 loc-prog net; 3 Ps for each loc-prog 7
- F0b: location-link's & matching-location -> loc-prog-name & not extend-net;
- F0c: apply-loc-prog & loc-prog-name & object -> loc-prog-result;
- F0d: get-loc-prog-compon & loc-prog-name -> loc-prog-compon's;
- F0e: extend-net & location-link's -> build-up-new-P & loc-prog-name;
- 7 file goals; 11 Ps 7
- F0f: file-goal -> trace-goal & recog-goal;
- F0g: recog-goal & matching-old-goal -> repeated;
- 7 F10's-F40's: file objects; 34 Ps; net: one P per distinguishing object feature 7
- F10a: file-object -> build-ext-repr & test-object-net & extend-object-net;
- F10b: 7 object net 7 test-object-net & sub-object-matches -> issame;
- F10c: extend-object-net & not issame
-> match-diff(using dummy object) & match-diff1 & match-result-exam;
- F10d: extend-object-net & issame -> match-diff & match-diff1 & match-result-exam;
- F20a: match-result-exam & match-result1 -> split-object-net-P-using-result;
- F40a: match-result-exam & not match-result1 -> replace-occurrences(with old object);
- 7 F50's: file desired assignments; 9 Ps; net: one P per desired assignment 7
- F50a: file-des-asg & assigns-n's -> assigns-n's & extend-des-asg-net;
- F50b: 7 des-asg net 7 assigns-n's & matching-old-des-asg -> issame-des-asg;
- F50c: issame-des-asg -> replace-occurrences(with old des-asg);
- F50d: extend-des-asg-net & not issame-des-asg -> extend-des-asg-net-actual;

des-asg = desired assignment.

Figure C.3 APs for the four kinds of filing

The goal-recognition process (APs F0f and F0g, or 11 Ps in GPSR, F6-F9N) has separate tests for each of the three types of goal (transform, reduce, and apply). The tests for apply goals are the most complex since they may or may not have been constructed with already-given operator differences. Two apply goals that are not given differences and that are otherwise identical would be the same, since each would find all such differences and try to reduce them. But the repetition of apply goals that are given differences are repeated based on the exact difference. As a side-effect of recognizing a goal, it is traced externally via "trace-goal" (F0f), which is used by the V's.

The object-filing process is based on an EPAM-like (Feigenbaum, 1963) discrimination. The object net consists of Ps that recognize isolated features of objects

• See Chapter III of this thesis, which contains other references and a more thorough discussion. The object-filing process is actually an advance in EPAM design over Chapter III.

and, on recognition, emit the name of an old object that looks like the object being filed on the basis of those features. When a candidate is proposed by the net to be similar to a new object, the filing process uses Match-Diff to verify the match or to find a difference between the proposed object and the new one. (Actually a variant of Match-Diff is used via the "match-diff1" signal, so that the match stops before finding all differences, since only one is necessary). A dummy object is used to extract an arbitrary feature from an object, if no existing object-net Ps match on any of the features of an object. When a difference is found, the P that proposed the candidate old object is split into two, one to continue to recognize features of the old object, and the other to do the same for the new one. The split actually consists of extending the old P in two mutually exclusive ways by adding conditions, or their negation, corresponding to the difference produced by Match-Diff. Thus a new object may cause more than one such split, since different features of it may match features of several old objects. The potential danger of too much net growth was not in practice a problem, for the tasks given to GPSR. A refinement to allow better selection of differences for making maximal distinctions between objects might be to use the DIFF:ORDER object, because perhaps easier differences are more likely to change and thus offer a better chance of providing a discrimination. Object-filing occurs at the beginning of a problem when the initial and desired objects are filed (QI) and it happens when the Try-Apply submethod receives the result of an operator application (M40b, from QA's).

The filing of desired assignments, the F50 APs, is like the loc-prog filing, based on complete information on the assignments. Only one type of P is constructed; when a new desired assignment is recognized to be an old one, all data items mentioning the new one are fixed up to refer instead to the old one. Filing and recognition of desired assignments occurs when they are constructed by the desirability selection process (M30c).

The method Ps are divided into the method-selection process, the three methods, and the Try-Apply submethod. The five AP groups, Figure C.4, correspond to the VAPs in Figure B.5 and to 64 GPSR Ps. The method selection process is always evoked by the executive, using the "select-method" signal, and the method selection in turn passes control to specific methods by specific signals. The M20 APs give enough detail on the Transform method to show the evocation of the Match-Diff submethod, the difference-evaluation routine (M20c), and the collection of results (by repeated P applications) from the Match-Diff submethod. The full detail of the use of the "transf-2" signal is not shown: there is a P firing that uses "transf-2" and emits another signal to allow the selection of the most difficult difference in M20e. This delay allows all match results to be completed before proceeding, and it is achieved using Psnlst's :SMPX to hold off examining the "transf-2" signal. M20e is one place where a sequence of two goals is set up, a reduce followed by a transform; the "next-trans" signal communicates this to the executive (see AP E10a in Figure C.2). The Transform method passes control back to the executive with a "succeed" signal (AP M20b) or with "file-goal" and "eval-goal" (M20e).

The Reduce method is composed of the three M30 APs, plus M40g, h, and i. M30a shows the application of the difference loc-prog to the TABLE:CONN object to get the operator or operator set relevant to reducing the difference. By repeated applications of M30b and M30c, all desirable assignments are constructed (see also Section D.3). Control always passes to the Try-Apply submethod, after evocation of the desired-assignment filing process and after generation of feasible assignments from desirable

% M0's: method selection; 5 Ps %

M0a: select-method & goal-type -> transform-method OR reduce-method OR moveop-method;

% M20's: Transform method; 14 Ps %

M20a: transform-method & not objects-same-name -> match-diff & transf-2;

M20b: transform-method & objects-same-name -> succeed;

M20c: match-result -> diff-eval & use-diff-eval-result;

M20d: diff-eval-result & use-diff-eval-result -> match-val;

M20e: transf-2 & most-difficult-match-val

-> file-goal & eval-goal & create-reduce-goal & next-trans;

% M30's: Reduce method; 20 Ps %

M30a: reduce-method & has-diff & not retry -> apply-loc-prog(diff, TABLE:CONN) & select-op;

M30b: select-op & loc-prog-result & is-move-op -> select-des-asg;

M30c: select-des-asg & move-op-compon's & loc-prog-compon's(diff)

-> file-des-asg & assigns-n's & gen-feas-asg & try-apply;

% M40's: Try-Apply submethod; 19 Ps %

M40a: try-apply & arbitrary-new-feas-asg -> apply-op & apply-check;

M40b: apply-check & apply-result -> file-object & succeed;

M40c: apply-check & opr-diff -> diff-eval & try-opr-diff-setup;

M40d: try-opr-diff-setup & hardest-opr-diff-for-feas-asg

-> has-opr-diff-asg & try-apply-2;

M40e: try-apply-2 & all-apply-op's-tried & easiest-has-opr-diff-asg(all feasasgs)

-> try-apply-result;

M40f: try-apply-2 & not has-opr-diff-asg -> methods-exhausted & fail;

% these are really part of Reduce method: %

M40g: try-apply-result & is-reduce-goal

-> file-goal & eval-goal & create-new-apply-goal-for-result;

M40h: reduce-method & retry & has-new-feas-asg -> try-apply;

M40i: reduce-method & retry & not has-new-feas-asg -> try-apply-2;

% M50's: Move-operator method; 6 Ps %

M50a: moveop-method & has-opr-diff & not retry

-> file-goal & eval-goal & create-new-reduce-goal & next-apply;

M50b: moveop-method & not retry & not has-opr-diff -> gen-feas-asg & try-apply;

M50c: try-apply-result & is-apply-goal

-> file-goal & eval-goal & create-new-reduce-goal & next-apply;

M50d: moveop-method & retry & has-new-feas-asg -> try-apply;

M50e: moveop-method & retry & not has-new-feas-asg -> try-apply-2;

diff = difference, des-asg = desired assignment, feas-asg = feasible assignment.

Figure C.4 APs for the method Ps

assignments (M30c evokes the task-specific generator QF using "gen-feas-asg"). The M40g AP shows how control passes back to the executive in case operator differences must be reduced. M40h and i show how the method is restarted (retried) depending on its

previous status. Other exits from the Reduce method are direct from within the Try-Apply submethod.

The Try-Apply submethod (M40a-M40f) is evoked by the Reduce and Move-Operator methods, using "try-apply" or "try-apply-2" signals. It may be evoked in a retry situation, in which case unselected responses from past evocations are tried (M40h, M40i, M50d, M50e). It has sole responsibility for evoking task-specific operator applications (M40a evokes QA's or QD's), using feasible assignments generated at the end of the desirability selection process (M30c), and for testing the results of those (M40b and c). It returns control to a parent process by the "try-apply-result" signal (M40e) in case of operator differences, or it passes control directly to the executive with "succeed" or "fail" (M40b and M40f). Note that APs M40a through M40e are applied repeatedly until a success occurs or until all operator differences have been extracted, allowing a selection to be made for maximal progress. Each operator difference, produced by QD's, is processed by M40c, which evaluates it using the difference evaluation routine ("eval-diffr", the D APs). Each operator-feasible-assignment pair may produce several operator differences, the hardest of which is selected by M40d. When all operator-feasible-assignment pairs have been tried (assuming none succeeds and M40b doesn't have a chance), M40e selects the easiest of the set of hardest ones from M40d, and emits it as the result. If a success does occur, the partial state of the selection stays around in Working Memory for possible resumption (retry). If M40e emits a result, the set that the result was selected from also stays around for use under retry conditions.

The Move-operator method is similar to the Reduce method in its control characteristics. M50a and b have an effect similar to M30a, b, and c, while m50c, d, and e correspond almost exactly to M40g, h, and i. M50b is a second instance (cf. the reduce-transform sequence in AP M20e) of the occurrence of a goal-sequencing operation, using a "next-apply" signal: a sequence of a Reduce goal and an apply goal are sprouted to achieve the apply goal that is the subject of the method's evocation. The "next-apply" signal is used in the executive AP E10b.

2 K's: matching (K for compare); 11 Ps 2

K0a: match-diff(objects) & not match-restriction(node) -> match-diff(nodes);

K0b: match-diff(nodes) & nodes-correspond
-> match-diff(daughter nodes) OR match-ok(terminal nodes);

K0c: match-diff(nodes) & not nodes-correspond -> extract-location;

K0d: extract-location & link-path-to-top-of-object & not match-diff1
-> file-loc-prog & location-link's & setup-result;

K0e: setup-result & loc-prog-name -> match-result;

K0f: extract-location & link-path-to-top-of-object & match-diff1
-> match-result1 & location-link's;

Figure C.5 APs for the Match-Diff submethod

The matching Ps constitute the the Match-Diff submethod, Figure C.5, which is evoked by the object-filing process (F10c and F10d) and by the Transform method (M20a).

The match is done in two phases, descent into the two objects from the top node along corresponding link paths until matching terminals are reached or until a difference is found, and ascent from the location of the differences to the top, collecting the set of links to form a loc-prog that will be used to describe the differences. KOa, KOb, and KOc perform the descent. KOb is applied repeatedly until terminals are reached or until KOc applies at the nodes in question. KOc corresponds to five Ps which define the ways for a match to fail. KOd and KOe extract and name the difference found in case the full match is desired. KOf extracts the location of the difference but does not name it, if the "match-diff1" signal is present. In this case the match result (the link path) is used directly by the object-filing process to build a sequence of condition elements for discriminating the objects matched.

% T's: transformations; 23 Ps %
 Ts: apply-transformation & object -> transformed-object;

 % C's: copying objects; 4 Ps %
 Cs: copy-object & object -> copied-object;

 % D's: evaluate diffrs; 5 Ps %
 DOa: diffr-eval -> apply-loc-prog(diffr,DIFF:ORDER) & diffr-eval-res1;
 DOb: diffr-eval-res1 & loc-prog-result & difficulty-criteria -> diffr-eval-result;

 % V's: trace goals; 9 Ps %
 Vs: trace-goal & goal-attributes -> printed-message;

 % X's: build external representation of objects; 6 Ps %
 Xa: build-ext-repr & object-links-and-values -> external-repr-of-object;

Figure C.6 APs for low-level processes

Finally, we can briefly consider some of the lower-level processes in Figure C.6. The T's are evoked by task-specific operator-applying Ps (QA's), as are the C's. The T's perform the operations listed in Figure B.4. The C's copy an object by creating new node tokens and attaching the corresponding links and values. The D's are evoked by the Transform method (M20c) and by the Try-Apply submethod (M40c), to evaluate match differences and operator differences, respectively. The result is a numerical value that is 100 times the difficulty given in the DIFF:ORDER object (i.e., a location-dependent measure) plus a heuristic that weights the type of the difference. Presently two such heuristics are used, as dictated by the tasks performed: one adds a weight of 5 to a difference whose actual value is "UNDEF" and whose desired value is some other constant; the other adds the absolute value of the difference between actual and desired values, if they are numeric. The numbers that go into this difficulty computation were arbitrarily chosen to result in reasonable ranges of values for the tasks chosen. The V's are evoked by the goal-recognition process to print out a trace message giving goal-type-dependent information. The X's build a human-readable representation from internally-constructed new objects. These are evoked by the object-filing process.

C.3. Meanings of the GPSR predicates

This section gives, in alphabetical order, explanations of the predicates that constitute the actual Ps for GPSR (Appendix A). As a preamble to the list, the following is a set of pointers to groups of predicates that belong together in representing various entities.

Goals (common to all types): HASDIFFIC, HASACTUAL:OBJ, HASSUPER:GOAL, HASTRACE:LEVEL, ISSAME:GOAL (only conditionally present).

Transform goals: ISTRANSFORM:GOAL, HASDESIRED:OBJ, HASANTEC (sometimes), HASALT:DIFFR (optional).

Reduce goals: HASDIFFR, HASOP (used only for debugging).

Apply goals: HASDES:ASG, HASOP, HASOP:DIFFR (sometimes), HASANTEC (sometimes).

Objects: HASTOPNODE, LINKS, HASVAL, ISSAME, ISSAME:EQV, ISDUMMY, ISDESCRIBED:OBJ, HASEXTREPR, MATCH:RESTR.

Loc-progs: HASNAME, HASLINK, HASLP:COMPON, APPLY:LOC:PROG, GETLP:COMPON, HASEXTREPR.

Assignments: ASSIGNS, ASSIGNS:N, ASSIGNS:D, ISSAME:DA.

Move operators: HASMOVE:COMPON, ISMOVEOP, HASVAR, HASVAR:LINK, VAR:DOMAIN, CHANGES:VAL, ISSET, INSET.

Points where a trace message is printed can be found by using the entry for TRACING in the predicate cross-reference, Appendix C. Other aspects of the process can also be found in this way; for instance, all places where difficulties are assigned to goals deal with HASDIFFIC. Reading through the Ps will give further hints for groupings of predicates by meaning.

Types for the predicate arguments:

a	assignment	lp	loc-prog
d	difference	n	node in object
da	desired assignment	o	object
g	goal	op	operator
k	numeric value	v	variable for task operator
l	link	w, x, y, z	arbitrary.

ADD-LAST(x,n) n is the last added node in ADD LINK set x. (T)●

ADD-LINK(o,x,lk) add a link to o. all instances with the same x form a set of links l that gives a location; k values order the set, increasing away from the top of o; when a non-numeric k is reached by the link path, k is the value placed at the location reached. (T, QA)●●

ADDPROD(P) P x is a created one; this instance is asserted when ADDPROD is executed to add the P. (F)

APPLY:CHK(g,op,da,a) check the results of the application of op, in context of g, with assignments da and a. (M)

APPLY:DIFFR(lp,x1,x2,op) an attempt to apply op has resulted in a difference at lp, with x1 where the operator desired x2. (M, QE)

● The primary P group that uses a predicate will be given in parentheses after the predicate description.

●● References to Q Ps are to Q's in the MC task, variant MC1, unless otherwise noted; the MC Ps are the only task Ps given in the cross-reference (Appendix C).

APPLY:DIFFR:SETUP(d,op,x1,x2) set up for operator difference d, for op, values x1,x2. (QD, QE)

APPLY:LOC:PROG(lp,o) apply lp to o. (M, D, LA)@

APPLY:OP(op,a,o) apply op to o using a; op is a move operator. (QA, QD, M)

APPLY:OP2(op,a,o) signal the completion of the operator difference process. (QD in MC1 only)

APPLY:OPF(op,o) apply op to o; op is a form operator. (M)

APPLY:RESULT(op,o) o is the result of applying op. (M, QA)

ASSIGNS(a,v,x) a assigns x to be the value of v. (QA, QD, QF, M)

ASSIGNS:D(da,v,x) da assigns x to be the value of v. (Q, F, V)

ASSIGNS:N(da,v,x) da tentatively assigns x to be the value of v; this changes to ASSIGNS:D after the da is filed. (F, M, DA)

CHANGES:VAL(v) v changes the value (HASVAL) at some location; that is, when the operator using v is applied, a value is changed from one non-numeric value to another. (QI in MK, M)

CHECK:NUMV(da) check if da assigns any numeric variables, as part of the desirability selection process. (M)

CHECK:RETRY(g) check if g can be retried. (E)

CHECK:SAME(g1,g2) check if apply-goals g1 and g2 have the same HASOP:DIFFR instances, so that g1 would be a repetition of g2. (F)

CHECK:SELX(g,op) the desirability selection process can generate no desirable assignment for op within g; this simply records the condition; the program makes no use of it at present. (M)

CHOOSE:OLD:GOAL(g) g is an old goal that is retryable; this signal defines a set of such goals from which a selection is made. (E)

CHOOSE:OLD:OBJ(o) o is an object that is a candidate for selection under the New-Obj criterion. (E)

COL:ANET(da,k,x,y) collect the components of a desired-assignment-net P; k is a count of the components, used to generate unique variable names; x is the LHS of the P; y is a list that is keeping track of mutual exclusion conditions on variables in the LHS. (see F50 ff.)

COL:LPNET(lp,d,x,k,y1,y2,z) collect the components of two loc-prog Ps; lp is the name of the loc-prog; d is the difference whose location defines lp (see HASLINK); x is a list of the links in lp; k is a count of created variables for the P; y1 and y2 are LHSs for the apply and recognize loc-prog Ps being built; z is a list of the components in lp for use in the loc-prog component P. (see F1 ff.)

COLONET(x,d,k,y,z1,z2,o1,o2) collect the components necessary to split object-net P x in two to distinguish between a new object o1 and an old object o2; d gives the location where a difference has been found; values for o1 and o2 at d are z1 and z2; k is a counter for created variable names; y is the piece of LHS that will be used in the split. (see F10 ff.)

COPY:LAST(x,n) n is the last node visited in the COPY:LINK set x. (T)

COPY:LINK(o,x,(k) copy a value at some location in o; the set of instances with the same x determine a link path using t; the k's are numeric, ordering the set of links to be followed, except the last link's k is the value to be placed at the location. (T,QA)

COPY:OBJ(o1,o2) o1 is to become a copy of o2; the arguments become (n1,n2) during the process. (C,QA)

DECR:LAST(x,n) n is the last node visited in the DECR:LINK set x. (T)

DECR:LINK(o,x,(k) decrement a value in o at some location; the set of instances with the same x determines a set of links to be the link path; the set is ordered by the k's, except the k that specifies the value of the decrement, which is represented as a negative integer. (T, QA)

DIFFR:EVAL(lp,x1,x2) a difference at lp with values x1 and x2 is to be evaluated and assigned a numeric difficulty (D, M)

DIFFR:EVAL:RES1(lp,x1,x2) step one of the DIFFR:EVAL process. (D)

DIFFR:EVAL:RES2(lp,x1,x2) step two of the DIFFR:EVAL process. (D)

DIFFR:EVAL:RESULT(lp,x1,x2,k) k is the result of the DIFFR:EVAL process. (D,M)

ERASE:APP(g,a) erase the APPLY:OP and ASSIGNS signals for g on a. (M)

ERASE:CHOICES(x) erase the choices generated in the Try-Old-Goals process; x is a dummy. (E)

ERASE:CHOICES:O(x) erase the choices generated in the CHOOSE:OLD:OBJ method; x is a dummy. (E)

• DA, OB, LA, LN, and LC refer to created net Ps, see Appendix D.

ERASE-CS(g) erase the CHECK SAME signals for g. (F)
ERASE-CSP(g) ready to do ERASE-CS(g); this is present just in case no ERASE-CS signal is otherwise emitted. (F)
ERASE-LPC(x) erase the HASLP:COMPON data; x is a dummy. (M)
ERASE-MATCH-DIFF(x) erase all MATCH-DIFF signals; x is a dummy. (M)
ERASE-MOI(o1,o2) erase MATCH-DIFF signals for o1,o2. (F)
ERASE-ML1(d,o1,o2) erase LOC:EXTR instances for o1, o2. (F)
ERASE-MNI(d) erase HASLINK instances for d. (F)
ERASE-MRI(d,o1,o2) erase MATCH-RES1 instances for o1, o2. (F)
ERASE-MVAL(g,x) erase MATCH-VAL signals for g, x. (M)
ERASE-OBJ(o) erase o, destroying LINKS and HASVAL's; o is also a node. (F)
EVAL-GOAL(g,k) evaluate goal g; if its difficulty is higher than k, it's too difficult. (E, M, QI)
EXT-DANET(da,op) check whether to extend desired-assignment net by adding da; op is the related operator. (F)
EXT-DANET2(da) extend desired-assignment net by adding da. (F)
EXT-LPNET(d) check whether to extend loc:prog net by loc:prog for d. (F)
EXT-LPNET2(d) extend loc:prog net by loc:prog for d. (F)
EXT-ONET(x,d,y1,y2,o1,o2) extend the object net, splitting P x; the difference between o1 (new) and o2 (old) is located by HASLINK's of d; values at that location are y1 and y2. (F)
EXTREPR(o) build the external representation for o. (X, F)
FAIL(g) signal failure of g. (E, M)
FAILED(g) g has failed. (E)
FEASASG(op,da,g) generate a feasible assignment for op from da; context g. (M, QF, F)
FILE-DES-ASG(da,op) file da; op is the related operator. (M, F)
FILE-GOAL(g) file g. (M, E, F, QI)
FILE-LOC-PROG(d) file the loc:prog given by the HASLINK's for d. (F, QD, QE, K)
FILE-OBJECT(o) file o. (F, M, QI)
FORM2INPUT-METHOD(g) signal for the (unimplemented) two-input form operator method. (M)
FORMOP-METHOD(g) signal for the (unimplemented) form operator method. (M)
GENDES-ASG(g,op,da,x,lp,k) generate variable-value pairs (ASSIGN:N) for da, in context of g, for op; x is the particular component of op (HASMOVE:COMPON), lp is the location of the difference being reduced, and k, if non-zero, is the size of the (numeric) difference. (M)
GENDES-ASG2(g,op,da,x,lp,k) second stage of GENDES-ASG; carries on after a check for components of lp. (M)
GETLP-COMPON(lp) signal that evokes a created lp P that emits the HASLP:COMPON's for lp. (M, LC)
GPSRINIT(x) initlinkz for the run; x is a dummy. (E, QI)
HASACTUAL OBJ(g,o) g has actual object o. (M, E, F, QI, V)
HASALT-DIFFR(g,lp,k,x1,x2) g has alternative difference at lp, difficulty k, values x1 and x2; only applies to transform goals under the RETRY-TRANS option. (M, E)
HASANTEC(g1,g2) g1 has antecedent goal g2. (E, V)
HASDES-ASG(g,da) g has da. (M, F, V)
HASDESIRED OBJ(g,o) g has desired object o. (M, E, F, QI, V)
HASDIFFIC(g,k) g has difficulty k. (M, E, V)
HASDIFFR(g,lp,x1,x2) g has difference at lp, actual value x1 and desired value x2. (M, F, V)
HASEXTREPR(o,x) o has external representation x; o can also be an lp. (F, X, V)
HASLHS(x,k,y) object-net P x has LHS y; k is the number associated with the last variable created for it. (F, QB)
HASLINK(d,l,k) d has l; k orders the set of links so determined, with 0 at the link most distant from the top of an object. (K, QD, QE, F, LN)
HASLP-COMPON(lp,l) lp has component l. (F, M, LN, LC)
HASMOVE-COMPON(op,x,y1,y2) move operator op has component x, which at a location specified by the HASVAR's of x brings about a change specified by an old value y1 and a new value y2; for instance, y1 = LOW, y2 = HIGH, specifying a numeric increase. (M, QI)
HASNAME(d,lp) d has lp as its location, as computed from HASLINK's for d. (F, LN, QI, K)
HASNEWFEAS(g,op,a,da,k) g has a new feasible assignment a, based on da, for op; its desirability is k, with higher values more desirable. (M, QF)

HASNEWFEAS:ORD(g,op,a,da,k)	signal that the desirability k is to be computed, as it appears in HASNEWFEAS; the arguments are as for HASNEWFEAS. (OF)
HASOP(g,op)	g has op. (F, M, E, V)
HASOP:DIFFR(g,op,lp,x1,x2)	the operator difference for g is at lp, with actual value x1, desired value x2. (F,M)
HASOP:DIFFR:ASG(g,op,da,a,k,lp,x1,x2)	op is being attempted by g, with a generated from da; a difference has been found at lp, actual value x1, desired value x2; k gives the difficulty evaluation of the difference. (M)
HASREPR(n,x)	n has external representation x, intermediate in the external representation collection process (EXTREPR) for objects. (X)
HASSUPER:GOAL(g1,g2)	g1 has supergoal g2. (E, M, V, QI)
HASTOPNODE(o,n)	o has n as its top node. (K, T, C, QA, QD, LA, OB, QI, X, F, E)
HASTRACE:LEVEL(g,k)	g has trace (indentation) level k. (E, V)
HASVAL(n,x)	n has value x; n is a terminal node of some object, without daughter LINKS's. (K, T, C, QA, QD, LA, OB, F, X, QI)
HASVAR(x,v)	x, a component of a move operator (see HASMOVE:COMPON), has v as a specifier of one of the links of the location of the component's change. (QI, M)
HASVAR:LINK(v,l)	l is the link associated with v; v is of a special type of variable that changes a value, so that the value assigned to it is that change as opposed to the link locating some change, as is the case for other variables. (QI, M)
INCR:LAST(x,n)	n is the last node visited in the INCR:LINK set x. (T)
INCR:LINK(o,x,l,k)	like DECR:LINK, except the value is to be incremented. (T, QA)
INSET(op,x)	op is in operator set x. (M, QI of Monkey task)
IS2INPUT(op)	op is a two-input operator (unimplemented). (M)
ISAPPLY:GOAL(g)	g is an apply goal (F, M, E, V)
ISDESCRIBED:OBJ(o)	o is a described object. (M, QI and QK of Monkey task)
ISDUMMY(o)	o is a dummy object, used as a match against some object to determine an arbitrary difference for it, in the object filing process; it has a top node only. (F, EO)
ISFORMOP(op)	op is a form operator (unimplemented). (M)
ISMOVEOP(op)	op is a move operator. (M, QI)
ISREDUCE:GOAL(g)	g is a reduce goal (E, F, M, V)
ISSAME(o1,o2,x)	o1 is the same as o2, as determined by object-net P x (a partial determination at best). (F, OB)
ISSAME:DA(da1,da2)	da1 is the same as the previously-known desired assignment, da2. (F, DA)
ISSAME:EQV(o1,o2)	o1 is the same as the previously-known o2, so that occurrences of o1 are equivalently o2. (F)
ISSAME:GOAL(g1,g2)	g1 is the same as the previously-known g2. (F, E)
ISSET(x)	x is a set of operators, in a table of connections. (M, QI of Monkey task)
ISTRANSFORM:GOAL(g)	g is a transform goal (E, F, M, V, QI)
LAST:DANET(x)	x is the last desired-assignment net P added. (F, E)
LAST:LPNET(x)	x is the last loc-prog net P added. (F, E)
LAST:ONET(x)	x is the last object net P added. (F, E)
LINKS(l,n1,n2)	l is the link between n1 and n2 in some object; n1 is the parent node, being closer to the top node. (K, T, C, F, QA, QD, LA, OB, X, QI)
LOC:EXTR(n,d,k,x1,x2,o1,o2)	extract the location of the difference, named d, between o1 and o2, with respective values x1, x2; k is 0 for the terminal node of the difference, and increases towards the top node, thus usable for ordering the extracted location (see HASLINK). (K, F)
LOC:PROG:RESULT(l,o,x)	x is the result of applying l to o. (LA, M, D)
MATCH:DIF1(x)	x is a dummy argument; this signals that the MATCH:DIFF process need only return a single difference, to be used in building the object recognition net. (K, F)
MATCH:DIFF(n1,n2,o1,o2)	match o1 and o2. n1 and n2 are the current nodes being matched, except that at the top level they are the objects themselves. (K, F)
MATCH:RES1(d,x1,x2,o1,o2)	MATCH:DIFF, as restricted by MATCH:DIF1, on o1 and o2, has found the difference d with respective values x1 and x2. (K, F)
MATCH:RESEXAM(o1,o2,x)	examine the results of matching o1 and o2, in the process of creating the object recognition net, x is the P that indicated o1 and o2 to be similar. (F)

MATCH:RESTR(l)	matches are not to examine branches of an object with l directly under the top node. (K, QI)
MATCH:RESULT(lp,x1,x2)	the result of MATCH:DIFF is a difference at lp, values x1 and x2. (K, M)
MATCH:VAL(lp,k,x1,x2)	the evaluation of MATCH:RESULT(lp,x1,x2) is k. (M, E)
MATCH:VSET(lp,x1,x2)	set up to use the result of DIFF: EVAL(lp,x1,x2). (M)
MCINIT(x)	initialize for Missionaries and Cannibals problem; cf. MKINIT and THINIT for the other two tasks. (QI)
METHODS:EXH(g)	methods for attaining g are exhausted. (E, M)
MORE:DA(x)	check for further signals to extend the desired-assignment net; this is necessary in case more than one assignment is to be filed at the same time. (F)
MOVEOP:METHOD(g)	use the Move-Operator method to attain g. (M)
NEXT:GOAL:APPLY(g,op,da,k)	the next goal to be tried after g succeeds is an apply goal, with op, da, and difficulty k. (M, E)
NEXT:GOAL:TRANS(g,o)	the next goal to be tried after g succeeds is a transform goal with desired object o. (M, E)
ONET:SUCG(g,o1,o2)	proceed from the object-net building process with o1 as the result of the success of g; o2 is the new object found to be the same as o1. (F)
ONET:SUCCH(g,o1,o2)	hold the emission of ONET:SUCG(g,o1,o2), allowing other processing to intervene (the presence of ONET:SUCG would interfere with that processing). (F)
RECOG:GOAL(g)	apply the recognition test to a new goal g, to see if it's a repetition. (F, E)
REDUCE:METHOD(g)	use the Reduce method to attain g. (M)
REDUCE:OPCHK(g,op,o)	check the result of applying op (a form operator) to o, in context g (unimplemented). (M)
REM:LAST(x,n)	n is the last node visited in the REM:LINK set x. (T)
REM:LINK(o,x,l)	remove some link or set of links from o; all such with the same x constitute a link path; the removal is of everything below the node at the end of the link path. (T, QA)
REPLHSP(x)	the LHS of P x has been replaced. (F)
REPRHSP(x)	the RHS of P x has been replaced. (F)
RESULT:SETUP(d,x1,x2)	set up to use the result of filing the loc-prog specified by d. (K)
RETRY(g)	g is being retried. (M, E)
RETRY:TRANS(x)	signal that the retry of transform goals is enabled; this is specified by the user at the start of a run, not internally according to some problem-solving strategy. (M, E)
SELECT:DES:ASG(op,lp,g,x1,x2)	select a desired assignment for op, to make a change from value x1 to value x2 at the location specified by lp. (M)
SELECT:METHOD(g)	select a method to apply to attain g. (M, E)
SELECT:NEW:OBJ(x)	signal that the selection of new objects (the New-Obj criterion) is enabled, as in GPS's Expanded-Transform method; this is an external switch, like RETRY:TRANS. (E)
SELECT:OP(g,x1,x2)	select an operator to change from value x1 to x2, in g. (M)
SPLIT:OB(w,x1,x2,y1,y2,z1,z2)	complete the splitting process on object-net P w; x1 and x2 are pieces of the LHS of w; y1 and y2 are values that determine how it's to be done; z1 and z2 are relevant variables within the P. (F)
SPROUT:RED:APP(g,op,lp,x1,x2,k)	sprout a new reduce goal with difference at lp, actual value x1 and desired value x2, to be followed by an apply goal, with op; k is a difficulty value to be used in evaluating the new goals. (M)
SPROUT:RED:TRANS(lp,k,x1,x2,g)	similar to SPROUT:RED:APP, except the goal to follow is a transform goal. (M)
SUCCEED(g,o)	g succeeds with result o. (M, E, F)
SUCCEEDED(g,o)	g has succeeded with result o. (E)
TEST:ONET(o)	test for the presence of objects similar to o in the object net. (F, OB)
TEST:ONETF(o)	finished with the test signalled by TEST:ONET(o). (F)
TEST:ONETR(o)	check the result of TEST:ONET(o). (F)
TEST:ONETS(o1,o2,x)	o2 was found to be similar to o1, as tested by P x. (F)
TRACE:ASG(da)	print a trace of da. (V)
TRACE:GOAL(g)	print a trace of g. (V, F)
TRACE:IND(k)	k is the trace indentation. (V, E, F, QA)
TRACE:OBJ(o)	print a trace of o. (V)

TRACING(x) a dummy predicate whose argument is a function call whose evaluation results in a printout of a trace line. (V, E, F, QA)
TRANSF2(g) signal the end of the first "step" of the transform method; the first "step" evokes the MATCH-DIFF method. (M)
TRANSF3(g) signal the end of the second "step" of the transform method, which evaluates MATCH-DIFF results; ready to proceed using those results. (M, E)
TRANSFORM METHOD(g) use the Transform method to attain g. (M)
TRY-OLD-GOALS(g) evoke the Try-Old-Goals process after quitting g. (E)
TRYAPP(g,op) evoke the Try-Apply submethod, to apply op to attain g. (M)
TRYAPP2(g) signal the final selection in the Try-Apply submethod. (M)
TRYAPP-DIFFR-SETUP(g,op,da,a,lp,x1,x2) set up to examine the results of evaluating an operator difference within the Try-Apply submethod; the arguments give the goal context (g,op,da,a) and the difference (at lp, actual value x1 and desired value x2). (M)
TRYAPP-RESULT(g,op,da,a,k,lp,x1,x2) the result of the Try-Apply submethod is a difference (at lp, actual value x1 and desired value x1) with difficulty k; g, op, da, and a are context. (M)
TRYAPPH(g,op) hold the TRYAPP signal for g and op while some other TRYAPP signal is being processed, or while multiple feasible assignments are generated. (M)
VAR-DOMAIN(v,x) x is in the domain of v. (M, QF, QI)
XRCOLL(n1,n2,l) collect the external representation (HASEXTREPR) for n1, in some object; n2 is the parent node of n1, linked by l. (X)

D. Tasks Given to GPSR

D.1. Justification of choices

GPSR has performed three of the eleven tasks given to Ernst and Newell's GPS. The three chosen ones represent a relatively wide range of variation, with the easiest one chosen as a good initial-debugging task. The easiest task for GPSR is the Tower of Hanoi (TH), in the sense of requiring a minimal amount of executive and method machinery. The next task on which GPSR was tested is the Missionaries and Cannibals (MC); this is the most difficult, requiring the full generality of the backup machinery in the executive and methods. The Monkey (MK) task was tested last; it has some important differences in terms of peculiarities of formulation.

Figure D.1 gives a hierarchy of the eleven GPS tasks, computed on the basis of Figure 90 of Ernst and Newell (1969, page 270), which gives the basic processes and methods used by GPS to solve the various tasks.

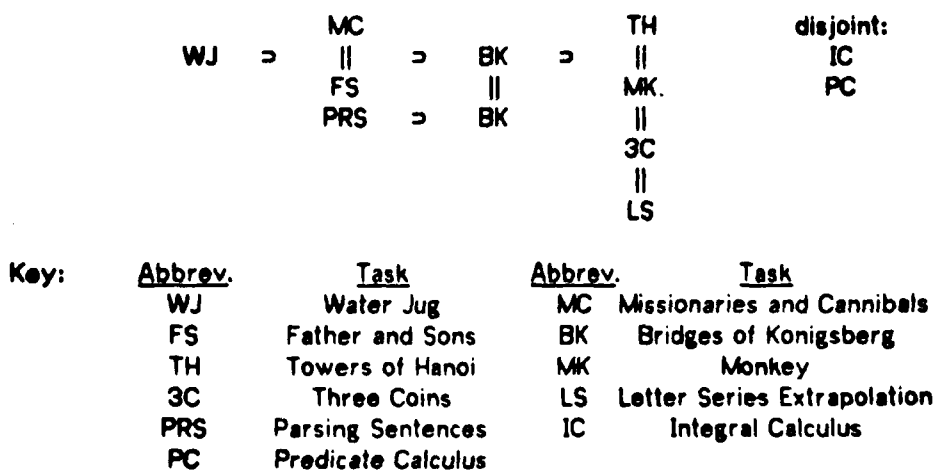


Figure D.1 A hierarchy of tasks by method usage

WJ uses ten methods, nine of which are used by MC; of MC's nine, six are used by BK and five of those are used in TH, MK, 3C, and LS. PRS uses the methods of BK, plus two others, but the two it uses are not used in MC, so PRS is on a distinct branch of the hierarchy. PC uses three methods not used by IC, and IC uses one not used by PC. PRS, PC, and IC differ from the other tasks in using form operators instead of using move operators exclusively. (GPSR has no facility for form operators; see Section G.1.) From the figure it is evident that the set of tasks chosen represents a suitable range of difficulty with respect to methods and processes used.

By peculiarities of task, beyond the method-process distinctions above, TH is

relatively simple both in specification and in behavior. MC is more complicated: it uses the capability of retrying transform goals, it has selection by the New-Obj criterion, goals fail and retrying is necessary, it has numeric differences, it has a match restriction, and its operator has post-tests. MK adds the following peculiarities: more than one operator, a non-restrictive table of connections, and a described object as its desired object. The three tasks TH, MC, and MK together have operators that use almost a complete set of transformations and tests (Figure B.4). In terms of problem-solving effort, MC is one of the most difficult of the eleven GPS tasks, whereas MK is one of the easiest; TH is intermediate, but its solution is done by GPS and GPSR with no mistakes in move choices.

D.2. The external representation of tasks for GPSR

APs for typical task Ps are given in Figure D.2.

```

% QI: initialize; 1 P %
QI:  init-signal -> gpsr-init & file-object(INITIAL:OBJECT and DESIRED:OBJECT)
      & file-goal & eval-goal & create-top-goal & INITIAL:OBJECT
      & DESIRED:OBJECT & TABLE:CONN & DIFF:ORDER & match-restriction's
      & var-domain's & move-op-compon's;

% QA's: apply operators; 1 P per operator %
QA's: apply-op & feas-asg & pre-tests-ok & post-tests-ok & transformations-feasible
      -> copy-object & apply-transformation's & apply-result;

% QD's: operator differences; 1 P per type of difference %
QD's: apply-op & feas-asg & test-bad -> file-loc-prog & opr-diffr-setup & location-link's;

% QE's: collect results of naming operator differences; 2 Ps %
QEa:  opr-diffr-setup & loc-prog-name & location-link's(some other diffr)
      -> file-loc-prog & opr-diffr;
QEb:  opr-diffr-setup & loc-prog-name & not location-link's -> opr-diffr;

% QF's: generate feasible assignments; 1 P per type of assignment %
QF's: gen-feas-asg & var-domain's & desired-asg-> feas-asg;

```

where opr = operator, diffr = difference, feas-asg = feasible assignment.

Figure D.2 APs for task-specific information

The task Ps are grouped into five types: QI does initialization, the QA's apply operators, the QD's extract operator differences, the QE's collect the results of operator difference filing (naming), and the QF's generate feasible assignments given desired assignments. The QI P initializes by setting up the main goal, defining task objects, and describing the move operators in a form usable by the desirable-assignment selection process (see M30c, Figure C.4); QI is evoked by an initialization signal typed externally by the user to begin the run. It evokes the executive initialization P and starts the solution process, with

signals used by E0a and E0b (see Figure C.2). The QA's apply task operators in response to "apply-op" from the Try-Apply submethod (M40a in Figure C.4), using feasible assignments generated by the QF's.

Operator differences are produced by the QD's, responding to the same signals as the QA's; there is usually one QD to recognize each type of operator difference possible, based on failure of individual tests that are necessary for operator application. Since many operator differences may be generated at once, the QE's are necessary to collect them and to make sure that all of them have been processed (their differences are filed and named) before passing control back to the Try-Apply submethod via the "opr-diffr" signal. The QF's respond to the "gen-feas-asg" signal from M30c and M50b in the Reduce and Move-operator methods (Figure C.4). The feasible assignment is generated using task-specific variable-domain information, and control simply falls back to the evoking process. The following subsections will go into more detail on how the objects, operator information, tests, and variable-domain information are expressed in particular tasks.

D.3. The Monkey task

The Monkey task (MK) has already been introduced in Section C.1. Two variants of MK were tried, one with the original Ernst and Newell table of connections, and a second with a more restricted one. The Ps for MK are at the beginning of Appendix B. Traces of the behavior for the two versions are in Appendix E and Appendix F.

The Ps for MK are the simplest of the Ps for the tasks. Q1 is the initialization P, of which representative extracts appear in Figure D.3.

The initial object:

& OBJECT('INITIAL OBJECT, (MONKEY PLACE PLACE1 BOX PLACE PLACE2))

where OBJECT is a PSMACRO that converts its second argument into a set of LINKS, HASVAL, and HASTOPNODE conjuncts.

The table of connections (more restricted variant):

& OBJECT('TABLE CONN, (MONKEY PLACE OP SET1 MONKEY HAND GET BANANAS BOX PLACE MOVE BOX))
& ISSET('OP SET1) & INSET('CLIMB, 'OP SET1) & INSET('MOVE BOX, 'OP SET1) & INSET('WALK, 'OP SET1)

Ordering the differences:

& OBJECT('DIFF ORDER, (MONKEY PLACE 1 MONKEY HAND 3 BOX PLACE 2))

Variable domains, used in constructing assignments

& VAR DOMAIN('MOVE TO, 'PLACE1) & VAR DOMAIN('MOVE TO, 'PLACE2)
& VAR DOMAIN('MOVE TO, 'UNDER BANANAS)

Operator components, used by the desirability selection process:

& HASMOVE COMPON('MOVE BOX, 'MB1, 'ARB, 'ARB) & HASVAR('MB1, 'MOVE TO)

Figure D.3 Extracts from the RHS of Q1 of the MK task Ps

The desired object is a described object, not given explicitly (to do so would be to give away the solution), but encoded as P QK, which recognizes and emits the critical difference between a given object and the desired solution; the critical difference is the absence of bananas in the monkey's hand. Two different tables of connections (TABLE:CONN) are given in Q1, one for each variant; the one selected depends on the positioning of the %

comment characters. Both of them make connection to named sets of operators, which are defined after the tables. After that, the DIFF:ORDER object and information on the components and variables of the move-operators is given.

To illustrate the representation of operator pre-tests, consider P QAM, the one to apply the operator MOVE:BOX (Figure D.4).

Respond to APPLY:OP signal with the appropriate argument:
 QAM: "APPLY MOVE:BOX" :: APPLY:OP(OP,A,OBJ) & SATISFIES(OP,OP EQ 'MOVE:BOX)

Bind assignment:
 & ASSIGNS(A,MT0,LOC)

Get the value of the monkey's location:
 & HASTOPNODE(OBJ,N1) & LINKS(L1,N1,N2) & SATISFIES(L1,L1 EQ 'MONKEY:PLACE) & HASVAL(N2,V1)

Check that the value of the box's location is the same, bound to V1:
 & LINKS(L2,N1,N3) & SATISFIES(L2,L2 EQ 'BOX:PLACE) & HASVAL(N3,V1)

Check that it's not already there and establish trace indent value:
 & VNEQ(V1,LOC) & TRACE IND(K)

Create new object token and print trace message:
 => EXISTS(O) & TRACING(TRACEPRINTM(<'APPLY,OP,TO,OBJ,GET,O,<MT0,LOC>>,K))

Signal that the object is to be copied:
 & COPY:OBJ(O,OBJ)

Set up the data for the COPY LINK operation, which changes values at two locations:
 & COPY:LINK(O,1,L1,LOC) & COPY:LINK(O,2,L2,LOC)

Report results, and erase the apply signal and the assignment:
 & APPLY:RESULT(OP,O) & NEGATE(1,3)

The operator difference P, QDM, is similar except:

Bind the assignment:
 & ASSIGNS(A,X1,X2)

Check that the box's location is different:
 & HASVAL(N3,V2) & VNEQ(V1,V2)

Check that the box isn't already at the desired location:
 & VNEQ(X2,V2)

Create a difference token, file the difference, and set up to report the result:
 => EXISTS(D) & FILE LOC PROG(D) & APPLY:DIFFER:SETUP(D,OP,V1,V2)
 & HASLINK(D,L1,O) & NEGATE(1,3)

Figure D.4 Extracts of operator application and operator difference Ps

MOVE:BOX has a variable MOVE:TO that gives the target location of the move. The pre-test for MOVE:BOX requires that both monkey and box be at the same place, a place not equal to the target location. Thus the test consists of finding the two locations and testing that they're the same, but distinct from the target; the fourth to the eleventh conjuncts in the LHS of QAM do just that, following links in the object from its top node to the terminal values, the identity being tested by simple match-variable identity (V1). In QDM, which generates an operator difference for MOVE:BOX, the corresponding test is for inequality of the two values, but otherwise the mechanics of the test are the same as in QAM.

The trace for MK given in Appendix E is accompanied by more detail than is given for any of the other traces: it includes the trace of P firings and a control flow summary that was generated from the P-firing trace. The P-firing trace is paragraphed, with breaks

occurring at places where a main goal trace line is printed in the ordinary trace (such a line starts with a series of dots). With this, the reader should be able to follow in great detail the workings of the program, at selected points in the trace. Each P firing as traced consists of a P name followed by "-", followed by that firing's ordinal number. The control flow summary is a graphic display of how control passes from one group of Ps to another; the groups are determined by P names' first letter. In addition, there is an indication of how many firings there were in a group before control changed to another group.

The behavior exhibited on MK with the original table of connections is interesting and useful for illustrative purposes, but is pathological for reasons explained below. The monkey walks to the box, climbs onto it, climbs down, climbs again, and finds itself in a repeated situation (G-11); going back to its starting place, it walks to the place under the bananas, finds that doesn't work, walks to the box, pushes it under the bananas, climbs, and gets the bananas. One interesting thing about the task formulation is brought out: there is no "unclimb" operator, but the WALK operator is sufficiently loosely specified that it serves the same function. The behavior is interesting because it illustrates some failure and backup. Its lack of direction is due to defects in the desirability selection process, whose task is to construct a partial assignment potentially suitable for reducing a difference.

The process of constructing an assignment is in three stages: selecting relevant operators based on the table of connections, constructing a desired (partial) assignment based on knowledge of the move operators, and constructing a feasible assignment from that, using information on allowable variable domains and restrictions. In this task, the move operators contain COPY:LINK operations, which simply change the value at some location in an object. The second stage makes use of operation-specific information, and in the case of MK, it was extended to take the COPY:LINK operations into account. That extension turned out to be inadequate, but it wasn't discovered to be so because of incorrect error diagnosis. The problem of the aimless behavior described above was diagnosed as resting in the first, relevant-operator selection, stage (it was thought that GPS simply had a lucky order in the way it picked operators) so that a more restrictive TABLE:CONN was constructed; the observed behavior then corresponded to the solution exhibited by Ernst and Newell (see the trace of the second version, Appendix F). The code for COPY:LINK operations is not selective enough because it doesn't take into account all the available (or potentially available) location information; for instance, CLIMB was selected in G-2 as desirable because no check was made to find the MONKEY:HAND location of the difference.

How the desirability selection would need to be changed to treat this task more intelligently involves two corrections: making the task specification slightly more informative, which can be done quite easily in the present framework; and reorganizing the process to react more smoothly to an abortive attempt to construct a partial assignment, and to be able to better detect when a partial assignment that has been built is as complete as possible within given information. These two corrections are sketched below, but have not been made for the present report because the correct diagnosis was slow in emerging, and because it is not sufficiently important to justify the effort involved (all tasks use parts of the process that would be affected). Also, the negative effects of the deficiency are local to the MK task.

The desirability selection process was not described in detail in Ernst and Newell, so that the approach for GPSR was to build it up to a level adequate for each task as it was tried, and to delay a more general treatment until it was found to be necessary. The mistake in developing the desirability selection process piecemeal as tasks were added to GPSR's repertoire was that for each task the tendency was to consider that everything done by the desirability selection process was evident in the result, namely that it only used information indicated explicitly in the desirable assignments to operator variables. This strategy is correct for TH because assigning variables is all that's necessary. It is not quite enough for MC, and in fact a patch was made to correct the deficiency there (the addition of HASVAR:LINK to associate a location link with the value assigned to a variable). The necessity of this patch was not transferred to MK, which is slightly worse because some operators have no variables at all. Instead, as described above, the fault was deemed to lie in TABLE:CONN, and fixing that made the behavior adequate.

It is now apparent that desirability selection must use full information about the locations of the effects of a move operator. The process is given the loc-prog of a difference and the desired change at that location. It must match this to a move operator component, which must specify the location of the change brought about by that component of the operator and the nature of the change that it brings about. The change is expressed symbolically as a pair of values: in MC, (LOW, HIGH) is used to indicate a numeric increase; in TH, (UNDEF, YES) is used to indicate moving a YES from one place to another; and in MK, (ARB, ARB) is used to indicate (vaguely) a change from one value to another (perhaps ARB, in retrospect, should be replaced by the name of a variable domain or some other set). The meanings of these symbols are built into the desirability selection process; for instance, it knows how to match a difference pair (3, 1) to the move-operator component (HIGH, LOW). The process first matches a move-operator component's change pair with the difference pair, and then matches the respective location descriptions. The location of the difference is given as a constant link path, but the move-operator component's location is in general a path some of whose links are variables with specified domains. When a variable matches to a constant, the process constructs a pair that goes into the desired assignment that is the process's output. The change necessary in the present implementation is that the move-operator component location must be specified and matched in its entirety, including all constant links, some of which are not presently used. This change is sufficient in general, and in particular will remedy the MK problem and include as a special case the solution that was used for MC.

The matching should be set up so that a failure to match completely is noted, resulting in an orderly abandonment of the process. In addition, given the general capabilities of the match, it might be useful to rate a desirable assignment according to goodness of fit and specificity of the effect to the change expressed by the difference (for instance (UNDEF, YES) is more specific than (ARB, ARB)); such a rating would perhaps allow some discrimination among a set of alternative operators or desired assignments.

D.4. The Tower of Hanoi task

The Tower of Hanoi task (TH) has been described in Section A, Figure A.1. Ps for the task and the trace of GPSR solving it follow those for MK in Appendix B and Appendix F. The task Ps have several features of note. The P that applies the MOVE:DISK operator,

QA, contains two nested conjunctions (form: NOT(EXISTS ...)) that perform in one sequence of tests the pretests for rule legality; the first tests that no smaller disk is on top of the disk that is to be moved, while the second tests that no smaller disk is on the peg onto which the disk is to be moved (see Figure D.5).

QA: "APPLY MOVE:DISK" : APPLY(OP,ASG,OBJ) & SATISFIES(OP,OP EQ 'MOVE:DISK)
 Establish variable bindings:
 & ASSIGNS(ASG,TP,TPV) & SATISFIES(TP,TP EQ 'TO:PEG)
 & ASSIGNS(ASG,FP,FPV) & SATISFIES(FP,FP EQ 'FROM:PEG)
 & ASSIGNS(ASG,HP,HPV) & SATISFIES(HP,HP EQ 'OTHER:PEG)
 & ASSIGNS(ASG,D,DV) & SATISFIES(D,D EQ 'DISK)
 Test that the disk to be moved is at the FROM:PEG.
 & HASTOPNODE(OBJ,N1) & LINKS(FPV,N1,FN1) & LINKS(DV,FN1,FN2)
 & HASVAL(FN2,FV) & SATISFIES(FV,FV EQ 'YES)
 Test that no smaller disks are on top of the disk to be moved:
 & NOT(EXISTS(D1,N2,N3,N3V) & LINKS(FPV,N1,N2) & LINKS(D1,N2,N3) & SMALLER(D1,DV)
 & HASVAL(N3,N3V) & SATISFIES(N3V,N3V EQ 'YES))
 And test that no smaller disks are at the target peg:
 & NOT(EXISTS(D1,N2,N3,N3V) & LINKS(TPV,N1,N2) & LINKS(D1,N2,N3) & SMALLER(D1,DV)
 & HASVAL(N3,N3V) & SATISFIES(N3V,N3V EQ 'YES))

Figure D.5 Pre-tests for the MOVE:DISK operator in TH

Note that the size of disks is determined by explicit Working Memory items that give the binary "smaller" relation between all disks (see Q1). The P that generates operator differences, QD, incorporates the trick used in GPS, namely that an operator is inapplicable if all the disks smaller than the one to be moved are not on the other peg, the peg that is neither the FROM:PEG nor the TO:PEG. So QD tests the other peg, and for every smaller disk that isn't there, it emits an operator difference. Tests for all the smaller disks are done in one match, and the P fires "simultaneously" for all the differences found. QE and QE2 ensure that all the operator differences found are filed and named before passing control back to the Try-Apply submethod; QE tests that all are finished, and QE2 reasserts signals to the filing process that have not been processed. The feasible assignment generator, QF, is also set up to fire more than once, simultaneously generating a set of combinations of feasible variable assignments (Figure D.6).

QF: "FEASIBLE ASG GEN" FEASASG(OP,DA,G) & SATISFIES(OP,OP EQ 'MOVE:DISK)
 establish assignments made by desirability selection:
 & ASSIGNS(D,DA,VAR,VAL) & SATISFIES(VAR,VAR EQ 'DISK)
 & ASSIGNS(D,DA,VAR2,VAL2) & SATISFIES(VAR2,VAR2 EQ 'TO:PEG)
 get the unassigned part of the peg set and apply exclusions.
 & ISPEG(VAL3) & ISPEG(VAL4) & VNEQ(VAL2,VAL3) & VNEQ(VAL2,VAL4) & VNEQ(VAL3,VAL4)
 assert the constructed assignments, one per successful match
 -> EXISTS(A) & HASNEWFEAS(G,OP,A,DA,0) & ASSIGNS(A,VAR,VAL) & ASSIGNS(A,VAR2,VAL2)
 & ASSIGNS(A,'FROM:PEG,VAL3) & ASSIGNS(A,'OTHER:PEG,VAL4) & NEGATE(1),

Figure D.6 Generation of feasible assignments in TH

It starts out with two operator variables (DISK and TO:PEG) already assigned by the desirability selection process. Using the set of pegs, ISPEG, it then arbitrarily selects values to be assigned to the other operator variables (FROM:PEG and OTHER:PEG) and finally forces the selections to be non-overlapping, with distinct values assigned to each operator variable.

The behavior of GPSR on this task is remarkable in that no mistakes are made. This is due to the trick mentioned above of looking at the "other" peg to extract operator differences. For instance, in Figure D.7, the goal is to move DISK4 to PEG3 (cf. G-18).

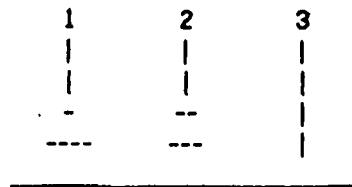


Figure D.7 Sample Tower of Hanoi situation

One approach would be to generate the operator difference that DISK1 should not be on PEG1; this would result in two alternatives to be tried in the search, since PEG1 can legally be moved either to PEG2 or PEG3 - doing the latter is definitely a mistake and cannot lead to a solution unless it is undone. The actual approach taken is to look at the other peg and note that DISK1 has to be there in order for the move to take place. This also results in two alternative attempted assignments, moving DISK1 to PEG2 from either PEG1 or PEG3, but the latter is easily rejected as not immediately applicable. By using the other-peg trick, GPS and GPSR avoid any backing up in the search.

D.5. The Missionaries and Cannibals task

The Missionaries and Cannibals problem (MC) consists of moving three missionaries and three cannibals across a river with a boat that will hold at most two people. The reason why this is a puzzle is that it must be done without allowing cannibals to outnumber missionaries on either bank of the river, for in this case the missionaries would be in grave danger (it is all right to have no missionaries). States in the problem are represented by objects that give the number of missionaries and the number of cannibals on each side of the river, plus the location of the boat. The GPSR task Ps and behavior traces are given after those for TH, in Appendix B and Appendix F. The task Ps for MC are given in two versions, which we'll refer to as MC0 and MC1 when it's necessary to distinguish. The basic structure of the two variants is the same, but they vary in the kinds of operator differences produced. This will be discussed further below.

An important feature of the GPSR representation of MC is how operator post-tests are implemented. In GPS, post-tests were a set of tests to be executed after the application of an operator, to ensure legality of the object. For MC, GPS had pre-tests to see that everything was legal on the from-side of the river before moving the boat across.

and it used post-tests to check legality of the to-side situation after making the move. Because of this structure, no operator differences were extracted as a result of failure of post-tests; the operator failed unconditionally. The GPSR formulation simply does away with post-tests by transforming algebraically the subjects of the post-tests into their corresponding pre-move values, thus allowing post-tests to be expressed as pre-tests (see Figure D.8).

First, QA establishes the following variable bindings:

M = the number of missionaries to be moved;

C = the number of cannibals to be moved;

NFM and NFC = the numbers of missionaries and cannibals on the from-side;

NTM and NTC = the numbers of missionaries and cannibals on the to-side.

Then QA tests that after the move, the from-side numbers are all right:

& NOT(SATISFIES2(NFM,M,NFM NEQ M) & SATISFIES3(NFM,M,NFC,NFM - M ?-LESS NFC - C))

And it tests the same for the to-side:

& NOT(SATISFIES2(NTM,M,0 NEQ NTM + M) & SATISFIES3(NTM,M,NTC,NTM + M ?-LESS NTC + C))

Figure D.8 Transformed post-tests for MC

Because of the simplicity of the operations that are in general available for use by move operators, this kind of inversion is always possible. Doing so allows operator differences to be obtained for the tests that were post-tests; obtaining differences from post-tests as GPS formulated them was not possible because such differences would be expressed in terms of a derived object rather than in terms of the object to which the operator was applied. The inversion of post-tests also implies that the difference produced when a "post-test" fails must also be inverted so that it is expressed in terms of the given object (see the RHSs of the QD's for details). Using inverted post-tests turned out to have an unexpected benefit: the necessity to retry transform goals (GPS's Expanded-Transform method), which was used by GPS in the MC task and in one other task, is no longer necessary for the MC task. An example of how this worked out will be pointed out when the behavior of GPSR on MC is discussed below. It is not evident that this kind of result holds more generally, and at present no way has been seen to attempt a formal proof.

GPSR's MC has a few other advantages over the GPS version. In the Ps that generate feasible assignments it was quite simple to add an ordering heuristic, to allow GPSR to pursue more sensible feasible assignments before less sensible ones. A more sensible assignment is one that moves two people across the river from the left bank to the right, which is the same direction as the overall goal, and that moves only one person on the return trip, from right to left; the less sensible ones do just the opposite. Notice that MC is challenging because it requires at one critical point a less sensible move. This heuristic was implemented by interposing a set of Ps to evaluate assignments (QF3 and QF4) between the assignment-generation Ps (QF and QF2) and the Ps that use the generated assignments (in the Try-Apply submethod). The ability to do this derives from the expression of the task as a PS program rather than as a passive data structure. Although it seems that such an idea would improve GPSR's performance to a large extent, the actual results are only a slight improvement, due to GPSR's tendency to go through a rather exhaustive search for this task.

Another place where GPSR has some advantages over GPS on the MC task is in the area of numeric differences and difficulties. The differences found by GPS (judging from its behavior traces) were simply to reduce, say, the number of cannibals on the left bank, whereas GPSR expresses it more precisely as reducing the number of cannibals from 3 to 0. This allows desired assignments generated by GPSR to assign values to variables in a way more precisely suited to the demands of a particular goal. At the same time GPSR's desirable assignments can be expressed as sets of values for some variable rather than a one-one value-variable correspondence. For instance, a desirable assignment might set the number of missionaries moved to be either one or two. This multiple value resulted in a natural way be allowing all possible matches in a desirability selection P to fire simultaneously; it is no problem for the feasible-assignment generator to use that kind of input for the same reason: it simply has more combinations to be matched and to allow to be simultaneously emitted. The result is a larger set of feasible assignments, but it contains all of the possibilities that might be relevant; this was obtained at no extra cost in terms of program code, since all the set maintenance is done automatically by the Pslist Working Memory and P examination stack, :SMPX. Finally, the more precise expression of differences allows the difference evaluation process to produce a more refined difficulty measure; for instance, reduction of a value from 3 to 0 is harder than a reduction from 2 to 0.

The difference between the task Ps for the MC0 and MC1 variants lies entirely in the way operator differences are extracted. In MC0 the operator differencing is set up in such a way that differences involving having the boat on the wrong side of the river to perform a desired move are found first (by P QDB), and other differences such as number of cannibals too high are found only if the boat is right. This variant was the first one tried, historically, because it corresponded with all the behavior visible in GPS's trace; that is, it seemed to be the case always that the boat difference would have to be reduced before other differences would be considered. But on reflection, the realization came that in fact since the numeric differences were considered to be harder to reduce (as established by DIFF:ORDER), it is incorrect to allow the boat difference to mask others. When GPS in general encounters a set of differences, it always attacks the hardest one first, since if that fails it is pointless to have wasted effort reducing an easy difference. As it turned out, this feature made some difference in the amount of progress GPSR could make in various experiments with MC. We will see below how this worked out, but at present we will point out some interesting points with respect to MC0 behavior, which compares more favorably with GPS's behavior than does the MC1 variant.

The first MC behavior exhibited (Appendix F) shows a successful run, in which GPSR solves MC (version MC0) correctly. In terms of number of goals, GPSR expends somewhat more effort than does GPS; GPSR has 69 goals where GPS had 57. But GPSR and GPS have some differences in what counts as a goal: in GPS, goals that are repetitions of past goals don't appear in the trace and are not counted whereas in GPSR, they do appear, as they are sprouted, so that the recognition and backup is more visible (this would decrease GPSR's count relative to GPS); on the other hand, direct applications of operators in GPSR (by the Try-Applly submethod) are not counted as goals, whereas in GPS they were apply goals that were generated and that immediately succeeded.

On the whole GPSR and GPS seem to approach the task in similar ways, and differences in their behavior is due mostly to the differences in the ways arbitrary choices

are made. But it must be pointed out that the GPS trace doesn't show two of the common types of behavior in the GPSR trace: GPS doesn't show any of the retrying of goals during the backup on failure; GPS doesn't show any instances of the Try-Old-Goals method. The GPS problem-solving executive was described as selecting the supergoal of a failed goal, and trying and exhausting other alternatives on that supergoal before backing up further; GPSR follows this with the resulting visible behavior, but corresponding behavior is not evident in GPS's traces. An example of this is the retry of G-36 after the failure of G-43; GPSR prints "RETRYING OLD" when a retry is due to selection by Try-Old-Goals in the executive. With respect to the lack in GPS of evidence of the Try-Old-Goals method, GPS's behavior is much more driven by the "New-Obj" selection, which chooses some newly-derived object as the subject of a transform goal rather than evoking the Try-Old-Goals method. GPSR also uses that kind of selection, but it uses it less often, simply because it doesn't generate new objects as rapidly as GPS did. On detailed comparison of behavior traces, it appears that GPS makes faster progress because it has less to work with: less refined difficulties and no differences from post-tests. This agrees with the general principle that too much information can be detrimental to a weak problem-solver. Some of the features that were introduced above as advantages have perhaps degraded behavior slightly because the problem-solver is not powerful enough to use the new material to advantage. We will see below some places where the means-ends methods can use extra supporting mechanisms.

There are three interesting features of GPSR's behavior on MCO (the third segment of Appendix F) that are worth pointing out. The first is that retrying transform goals is no longer essential for solution of the problem; this is evident in the behavior starting at goal G-36, Figure D.9.

```

SELECT BY NEW OBJ, O-13
. G-35 : TRANSFORM O-13 TO DESIRED OBJECT (FROM G-1)
O-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
.. G-36 : REDUCE 2 TO 0 AT (LEFT CAN) OF O-13 (DIFFIC 202) (FROM G-35)
  APPLY CROSS RIVER TO O-13 GET O-18 (FROM LEFT TO RIGHT NMIS 1 NCAN 1)
  O-18 IS THE SAME AS O-12

RETRYING G-36
... G-50 : APPLY CROSS RIVER TO O-13 (DIFFIC 202) (FROM G-36)
  ASSIGNS FROM ← LEFT, NCAN ← 1, NCAN ← 2
  O-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
.... G-51 : REDUCE 2 TO 0 AT (LEFT MIS) OF O-13 (DIFFIC 202) (FROM G-50)
  APPLY CROSS RIVER TO O-13 GET O-22 (FROM LEFT TO RIGHT NMIS 2 NCAN 0)
  G-51 SUCCEEDS
.... G-52 : APPLY CROSS RIVER TO O-22 (DIFFIC 202) (FROM G-50 AND G-51)
  ASSIGNS FROM ← LEFT, NCAN ← 1, NCAN ← 2
  O-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))

```

Figure D.9 GPSR doesn't need to retry transform goals for MC

This is a goal to reduce the number of cannibals on the left bank; it comes from a transform goal (G-35), and retrying the transform goal would ordinarily be thought necessary to try to reduce the number of missionaries instead. In the particular situation at G-35, it is in fact necessary to try the latter reduction, since the former cannot lead to the solution. GPSR's initial attempts on G-36 are abortive, but a little more than a column after the first try, G-36 is retried, and the subordinate goal G-51 is to reduce the number of missionaries, as required for solution. G-51 comes about as the result of an operator difference from an inverted post-test; GPSR tries to move one or two cannibals across the river to find that this results in too many cannibals there, so the subgoal to increase the number of missionaries on the right side (equivalently reduce the number on the left) is created as G-51. G-36 and G-51 are two alternatives that would be obtained by trying or retrying G-35, but here the latter was developed in the subgoal structure of the former.

The second feature of GPSR's behavior is that it must return to retry a goal that at its first occurrence looked too difficult to work on. The goal in question is G-14, generated first in the first column of the trace in the third segment of Appendix F, which is excerpted in Figure D.10.

```

.. G-8 : TRANSFORM O-3 TO DESIRED OBJECT (FROM G-3 AND G-4)
   O-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
... G-9 : REDUCE 2 TO 0 AT (LEFT CAN) OF O-3 (DIFFIC 202) (FROM G-8)

      G-9 SUCCEEDS
... G-13 : TRANSFORM O-5 TO DESIRED OBJECT (FROM G-8 AND G-9)
   O-5 (LEFT (CAN 0 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
... G-14 : REDUCE 3 TO 0 AT (LEFT MIS) OF O-5 (DIFFIC 203) (FROM G-13)
      NO PROGRESS, G-14 FAILED

      RETRYING OLD G-14
..... G-23 : APPLY CROSS RIVER TO O-5 (DIFFIC 105) (FROM G-14)
          ASSIGNS FROM ← LEFT, NMIS ← 1, NMIS ← 2

      G-23 SUCCEEDS
      G-14 SUCCEEDS
..... G-26 : TRANSFORM O-12 TO DESIRED OBJECT (FROM G-13 AND G-14)
   O-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))

```

Figure D.10 GPSR retries a no-progress goal

When it is generated, its difficulty is higher than that of its supergoal, so preceeding on it violates GPS's principle of working from the difficult to the not-so-difficult. GPSR abandons it to work first on some easier goals whose alternatives are not exhausted, but about a column later in the trace, it returns to retry G-14, and this leads to substantial progress. This feature surprised me when it occurred because the static goal-subgoal

structure violates the keep-progressing principle, although dynamically GPSR's behavior does not.

The third feature involves an instance where the basic means-ends strategy as implemented in GPSR appears fatally inflexible[•]. The critical segment of behavior starts at G-26 in the trace, and runs through G-30, Figure D.11.

```

.... G-26 : TRANSFORM O-12 TO DESIRED OBJECT (FROM G-13 AND G-14)
      O-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
..... G-27 : REDUCE 1 TO 0 AT (LEFT CAN) OF O-12 (DIFFIC 201) (FROM G-26)
..... G-28 : APPLY CROSS RIVER TO O-12 (DIFFIC 105) (FROM G-27)
      ASSIGNS FROM ← LEFT, NCAN ← 1
..... G-29 : REDUCE UNDEF TO YES AT (LEFT BOAT) OF O-12 (DIFFIC 105) (FROM G-28)
      APPLY CROSS RIVER TO O-12 GET O-13 (FROM RIGHT TO LEFT NMIS 1 NCAN 1)
      G-29 SUCCEEDS
..... G-30 : APPLY CROSS RIVER TO O-13 (DIFFIC 105) (FROM G-28 AND G-29)
      ASSIGNS FROM ← LEFT, NCAN ← 1
      O-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))

```

Figure D.11 GPSR carries over desired assignments inappropriately

GPSR sets up an apply goal with a desired assignment to move one cannibal from the left. To do this, it first reduces the operator difference of the wrong boat location, and in doing so adds a cannibal. The supergoal's desired assignment is carried over to the result of that reduction, according to normal operating procedure, but now it is no longer appropriate since a cannibal has been added. It turns out that the object that resulted from the intervening reduce goal is on the critical path to the solution; failing to treat it appropriately in this case is a fatal mistake because the context is such that that object is never again generated or examined. Fortunately, GPSR has the New-Obj selection criterion, by which derived unexamined objects are used in new transform goals. This method is not part of the pure means-ends strategy, but in this case it saves it from failure by proceeding with that intermediate result. It was surprising to discover such a fundamental weakness in the basic GPS methods; GPS's way around this difficulty was not originally presented as a patch to add completeness to means-ends.

The MC1 version differs from MC0 in having more flexible operator difference capabilities. QD Ps are the only case where it is necessary to generate several operator differences using more than one P, so it is necessary to add some control to ensure proper sequencing. (In MK, only a single difference is ever detected, and in TH, all the differences are generated simultaneously by a single P.) There are separate Ps to detect operator differences from a wrong boat location (QDB), from conditions at the from-side (QDF1, QDF2), and from conditions at the to-side (QDT1, QDT2). QDZ detects the completion of trying to apply the operator, where each QD re-asserts the apply signal to make sure other difference Ps get a chance. The QD's correspond roughly to the Ps of MC0, except that in MC0, QDF's and QDT's conditions included checks on the boat's location, so that boat differences would mask the others.

• GPS has similar properties. See goals 6-9 on page 132 of Ernst and Newell, and goals 18-21 on page 187.

A trace of the behavior of version MC1 is given after the one for MC0 discussed above, in Appendix F. This trace shows GPSR without the New-Obj selection criterion. GPSR fails to find the solution, due to features already discussed. This behavior is presented to demonstrate what results from the more refined operator-differencing Ps. The basic difference between MC1 and MC0 (whose behavior with these options is not shown) comes in the section of the trace between G-38 and G-61: one object, O-23, is obtained in the solution that isn't obtained by the MC0 version. However, this doesn't help it to get to the solution, because of the desirable assignment inflexibility; further, it incurs a large cost: 67 goals are generated, versus 38 in the corresponding MC0 version. The difference that is the subject of G-38 is the result of the revised Ps; it produces O-23 while retrying G-43, just before G-46 in the trace.

Finally with respect to MC, GPSR fails to solve the task without New-Obj even when the option is enabled that allows transform goals to be retried (that trace is not included). Three objects are not actual objects in transform goals, and one is the critical object necessary for solution, so that it is clear that New-Obj is required for GPSR as it stands. The goal-subgoal contexts in which those objects are generated is such that no further progress is possible. Furthermore, GPSR makes less progress than without the retry-transform option because the goal-subgoal structure is such that some of the branches in the search are blocked by repeated goals, that is, goals that are repeated because the option generated them where they weren't generated before; the "repeated" goals are not really repeated in the sense that they occur in different goal-subgoal contexts from the previous instances.

To summarize the results of doing the MC task, extracting operator differences from inverted post-tests removes the necessity for retrying transform goals. GPSR puts forth somewhat more effort in solving the MC task than did GPS, because of the extra information and refinement that GPSR has from its operator differences and difference evaluation. GPSR's behavior exhibits a basic inflexibility in the means-ends analysis structure that forces the inclusion of the New-Obj selection principle; this inflexibility is in passing desired assignments from a goal to its descendent. It is necessary to violate statically the principle of proceeding from the more difficult to the less difficult, at least with respect to GPSR's difficulty measures and difficulty propagation rules. A slight improvement in GPSR's achievements results from expanding the operator differences produced, but the cost is high in terms of extra goals generated. Various blocks to progress occur due to repetition of a single goal in different contexts, especially contexts where progress towards the solution might occur. Most importantly, the basic means-ends analysis strategy, with transform goals retryable, is incomplete, in the sense of being unable to reach a solution.

Further research to find the underlying reasons for the above faults, and to try variations in the organization of the problem-solving executive and methods are beyond the scope of the present work. Important features of PSs have been brought out regardless of problems with GPS itself. The MC task was solved using the mechanisms GPS used; the investigation of the defects exposed by GPSR is beyond that basic objective.

D.6. A comparison of task specification in GPS and GPSR

A task specification for GPS consisted of a list of list-structured elements with enough information for GPS to construct an adequate internal representation of a task. It started with several meta-components that influenced the interpretation of further components. Then came descriptions of the problem to be solved (the top goal), definitions of entities used in other components, and definitions of operators, of the table of connections, etc. The following table summarizes the differences between the external representation of tasks for GPS and GPSR.

<u>GPS</u>	<u>GPSR</u>
RENAME	Unused and unnecessary; GPSR has fewer internal naming conventions to be adhered to, and those are not subject to change.
SKIP-WORDS	Inapplicable.
DECLARE	No types, so unnecessary (no distinction between types of links in object representation; also no need to give types for FEATURES, SETS, etc.).
TOP-GOAL	List of instances in the RHS of QI; described objects are encoded as Ps (QK in MK).
FEATURES	Conversion and definition is automatically done by GPSR for loc-progs; other definition of structures is unnecessary.
SETS	Lists of instances in RHS of QI (Working Memory is a set).
EXPRES	Expressions encoded directly in Ps.
TESTS	Encoded directly in Ps (QA's, QD's); no separate definition.
TRANSFORMATIONS	Evoked by name by instances in RHSs of QA's; feasibility must be checked in LHSs of QA's.
MOVE-OPERATORS	Instances in RHS of QI (ISMOVEOP).
type of operator	Sets of instances (VAR:DOMAIN) in RHS of QI plus tests on those in the feasible assignment generators, QF's.
VAR-DOMAIN	Tests in LHSs of QA's and QD's.
PRETESTS	Tests in LHSs of QA's and QD's.
POST-TESTS	Evocations of transformations in RHSs of QA's plus HASMOVE:COMPON instances from RHS of QI.
MOVES	Encoded directly in Ps that respond to MATCH:DIFF signal.
DESCRIBED-OBJ	A list structure that is converted to the Working Memory representation of LINKS and HASVAL's by the OBJECT PSMacro.
OBJECT-SCHEMA	The DIFF:ORDER object, asserted by QI.
DIFF-ORDERING	The TABLE:CONN object, asserted by QI.
TABLE-OF-CONNECTIONS	Instances in RHS of QI that are used to restrict the Match-Diff submethod (MATCH:RESTR); otherwise full object comparisons are assumed.
COMPARE-OBJECTS	

Thus, GPSR uses the same basic set of task-specific components as GPS, except for the first three above. The components' organization is of course quite different, with

major departures in the representation of move operators. Move operators are encoded as Working Memory instances asserted by QI, for the desirability selection process, as Ps to generate feasible assignments given a desired assignment, as Ps to apply operators, and as Ps to produce operator differences. The rest of this subsection will review in detail some aspects of the encoding of move operators as Ps, with occasional references to other major differences.

A move operator consists of a sequence of tests for applicability followed by a sequence of transformations (REM:LINK, INCR:LINK, and others, see Figure B.4) that produce a new modified object from a copy of the input object. GPSR also performs a set of tests in LHSs of QA's to ensure that the transformations are legal. This was not done by GPS, which simply allowed the transformations to "fail", but it is important in GPSR to avoid the (unnecessary) extra work involved in doing and undoing the copying of objects.

MK has four very similar move operators, of which MOVE:BOX is typical, containing types of tests that represent the full variety in the four operators. MOVE:BOX has a VAR-DOMAIN,

X IS IN-THE-SET OF PLACES,

which is guaranteed in GPSR by the desirability selection process, using Working Memory instances asserted by QI:

VAR DOMAIN('MOVE TO,'PLACE1) & VAR DOMAIN('MOVE TO,'PLACE2) & VAR DOMAIN('MOVE TO,'UNDER BANANAS)
& HASMOVE COMPON('MOVE BOX,'MB1,'ARB,'ARB) & HASVAR('MB1,'MOVE TO)

MOVE:BOX has two PRETESTS. The first is a simple set membership test on the monkey's location, which is simply assumed to be true in GPSR, since it is redundant,

THE MONKEY'S-PLACE IS IN-THE-SET OF PLACES.

The second is

THE MONKEY'S-PLACE EQUALS THE BOX'S-PLACE,

which in QAM (the application P) becomes

LINKS(L1,N1,N2) & SATISFIES(L1,L1 EQ 'MONKEY PLACE) & HASVAL(N2,V1)
& LINKS(L2,N1,N3) & SATISFIES(L2,L2 EQ 'BOX PLACE) & HASVAL(N3,V1)

making the EQUALS test by using the same variable, V1. In QDM, which produces the operator difference, the test is similar except the last element becomes,

HASVAL(N3,V2) & VNEQ(V2,V1)

For MK, the desired object consists of a test that the monkey has the bananas; if that is not true, a difference is produced as a match result. The test is:

NOT(EXISTS(L,N2,V) & LINKS(L,N1,N2) & SATISFIES(L,L EQ 'MONKEY HAND)
& HASVAL(N2,V) & SATISFIES(V,V EQ 'BANANAS))

which corresponds to TEX-DESCRIPTION,

THE CONTENTS-OF-THE-MONKEY'S-HAND EQUALS BANANAS.

For TH, the move operator MOVE:DISK has in GPS four VAR-DOMAIN's and a PRE-TEST. Three of the former are of the form,

THE TO-PEG IS AN EXCLUSIVE-MEMBER OF THE PEGS

In GPSR, QF has the equivalent as:

SATISFIES(VAR2,VAR2 EQ 'TO PEG) & ASSIGNS D(VAR2,VAL2) & ISPEG(VAR2)
& VNEQ(VAR2,VAL3) & VNEQ(VAR2,VAL4),

where VAL3 and VAL4 are bound to other members of ISPEG that are used to construct a feasible assignment in the RHS. That is, the EXCLUSIVE-MEMBER relation is achieved by the VNEQ's in the LHS of the P. The fourth VAR-DOMAIN is for DISK, which is a set membership without exclusions and thus a simple variation of the above.

The PRE-TEST for MOVE-DISK in GPS is

X ON THE OTHER-PEG IS DEFINED FOR-ALL X SMALLER THAN THE PARTICULAR DISK

In GPSR, QA has

```
NOT( EXISTS(D1,N2,N3,N3V) & LINKS(FPV,N1,N2) & LINKS(D1,N2,N3) & SMALLER(D1,DV)
    & HASVAL(N3,N3V) & SATISFIES(N3V,N3V EQ 'YES') )
& NOT( EXISTS(D1,N2,N3,N3V) & LINKS(TPV,N1,N2) & LINKS(D1,N2,N3) & SMALLER(D1,DV)
    & HASVAL(N3,N3V) & SATISFIES(N3V,N3V EQ 'YES') )
```

where FPV is bound to the binding of FROM:PEG, TPV to the binding of TO:PEG. That is, rather than having a positive test for all disks on the OTHER peg, which is a variable number and thus impossible to express in a single P, the PRETEST is converted to its negative so that "FOR-ALL" becomes "EXISTS" and one P now suffices for the test. On the other hand, QD which generates operator differences uses the positive form of the PRETEST because it must generate a difference for each such disk; it does so by firing once for each smaller peg that is not on the other peg:

```
& NOT( EXISTS(V,N3) & LINKS(D1,N2,N3) & HASVAL(N3,V) & SATISFIES(V,V EQ 'YES') )
```

where D1 is bound to a disk smaller than the disk to be moved, and N2 is the node of the TH object that is linked from the top node by the value of OTHER:PEG.

For MC, the CROSS-RIVER operator's two POST-TESTS are:

1. ARE ANY OF THE FROM-SIDE-TESTS TRUE
2. ARE ANY OF THE TO-SIDE-TESTS TRUE

where

FROM-SIDE-TESTS = (

1. THE M OF THE FROM-SIDE IS NOT-LESS-THAN THE C OF THE FROM-SIDE.
2. THE M OF THE FROM-SIDE EQUALS 0)

and TO-SIDE-TESTS is similar, with TO-SIDE for FROM-SIDE. "TRUE" here is defined to be a disjunction, namely of the two elements of the FROM- and TO-SIDE-TESTS. The GPSR version of this has been given already, Figure D.8. Note that the negation of the negation of disjunction is used, converting it to a nested conjunction. In the operator difference case, QDF and QDT for the two types of differences, the simple negation of the disjunction is used, which is the two conjuncts in QDF,

```
SATISFIES2(NFM,M,M NEQ NFM) & SATISFIES3(NFM,M,NFC,NFM-M ?-LESS NFC-C)
```

where M is the number of missionaries to be moved, C the number of cannibals, NFM the number of missionaries on the from-side, and NFC the number of cannibals on the from-side. Recall that for POST-TESTS in GPSR, the conditions are algebraically reversed (Figure D.8). The only other feature in MC that hasn't been illustrated for the other tasks is the form of VAR-DOMAIN, which in GPS was

Y IS A CONSTRAINED-MEMBER OF THE 0,1,2-SET, THE CONSTRAINT IS X.Y IS IN-THE-SET 1,2

and similarly for X. As a domain restriction, this is encoded in GPSR in the feasible assignment generation P, QF2, as:

```
VAR DOMAIN(VAR3,VAL3) & SATISFIES(VAL3,NUMBERP VAL3)
& VAR DOMAIN(VAR4,VAL4) & SATISFIES(VAL4,NUMBERP VAL4)
& SATISFIES2(VAR3,VAR4,VAR3 LEXLT VAR4)
& SATISFIES2(VAL3,VAL4,VAL3+VAL4 ?-GREAT 0) & SATISFIES2(VAL3,VAL4,VAL3+VAL4 ?-LESS 3)
```

where VAR3 and VAR4 are bound to operator variables (corresponding to X and Y above) whose prospective values are bound to VAL3 and VAL4; the two SATISFIES ensure numeric domains for VAL3 and VAL4 (using data asserted by Q1 that defines the domains as the sets {0, 1, 2}), the first SATISFIES2 ensures that VAR3 and VAR4 are distinct, and the last two SATISFIES2's apply the sum constraint. Thus in this case the PS match has examined all possible assignments and eliminated the unwanted ones by applying the constraint (how GPS implemented this same function is unknown; there are of course alternate ways to express it in Psnlst).

There are several points to be made about the above comparisons. The direct encoding of the full variety of GPS tests as segments of LHSs of Ps was achieved within the PS language. The level of expression of the PS version is similar to the GPS external language, except that the PS version requires the use of local match variables in addition to the GPS operator variables. In several places the expressive power of Psnlst was useful in reversing the logical sense of tests and in algebraically transforming tests. The PS expressions are procedural in the sense of being used as programs, yet retain the declarative aspect of the GPS external language.

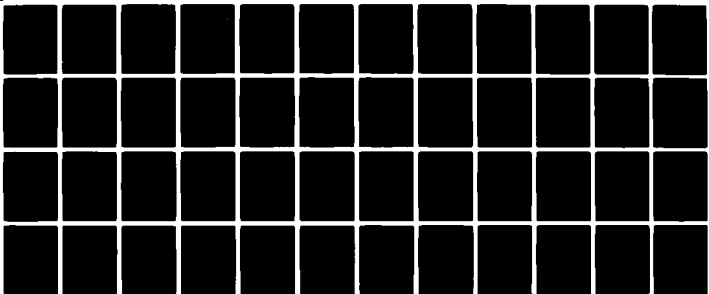
AD-A118 468

CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 9/2
PRODUCTION SYSTEMS AS A PROGRAMMING LANGUAGE FOR ARTIFICIAL INT--ETC(U)
DEC 76 M D RYCHENER F44620-73-C-0074
NL

UNCLASSIFIED

2-2

AL
CLASS



END

DATE

FILED

9-82

DTIC

E. Production-System-Related Features of GPSR

The work with GPSR has raised two categories of issues, those dealing with PS aspects and those dealing with GPS itself. PS issues deal with control and representation, and the position of GPSR with respect to these can be expected to remain, independent of variations in the GPS content of the program. The issues raised with respect to GPS suggest a number of further experiments, but these are beyond the present scope, so that they will simply be pointed out. This section considers PS issues, and the following section, the GPS issues. The final section points out problems for further research.

In Section E.1, we discuss features of the implementation that are related to PSs viewed as a programming language, namely implementation time, conciseness, control, and efficiency. In Section E.3, we have discussed the places in GPSR where PSs produce the greatest impact on representation. The power inherent in the PS match is used to advantage in making complex selections, in generating combinations of feasible assignments, and in accessing locations in objects. The flexibility of varying degrees of procedural encoding of knowledge in PSs will be discussed in Section E.2. The openness of control, the high level of PSs as a language, and the advantages of the explicitness of conditions and actions are supported by several aspects of GPSR. First, it was easy to make several extensions to the basic GPSR program, in extending its applicability to the three tasks. This will be discussed in detail in Section E.3. Second, task specifications were directly expressible as Ps and Working Memory items. The interaction of these Ps with the main program is minimal; there is the potential of having arbitrary PS programs being driven by GPSR (or vice versa). The process of encoding knowledge in the task Ps is the subject of Section E.4. Third, as experiments developed new knowledge of requirements for the problem-solving executive, it was straightforward to add the new knowledge and determine its interactions with other parts of the executive. This will be discussed in Section E.5.

E.1. Low-level implementation features

Implementing GPSR took about 260 hours, and in addition about 60 were spent in preparatory study. The programming time was split up roughly into 30% coding time and 70% debugging, which includes testing and making minor changes to the program. The reading was spread over about 7 weeks and the programming, over 23, in the sense that out of the total elapsed time of about a year, only about half of the weeks showed significant effort spent on GPSR.

Qualitatively, the GPSR program is fairly close to the abstract descriptions of GPS presented in this chapter and in the Ernst and Newell book (Section B.2 has a detailed comparison). The language is high-level and concise. GPSR has 206 Ps and 10 auxiliary (PSMacro) Lisp functions, which are involved with printing the behavior trace and converting objects between internal and external representations. The average number of task Ps is 11. This can be compared to data for a direct ancestor to the GPS program that GPSR imitates, GPS-2-2 (Newell, 1963). The primary structure missing from GPS-2-2 is the problem-solving executive (it also lacks a method language interpreter, but so does

GPSR). GPS-2-2 has 250 IPL-V routines, averaging around 20 words per routine. Since IPL-V takes a line for each instruction, and each instruction takes an IPL-V word, this gives an estimate for length of program listing at over 5000 lines, including data structures, but not including an estimate of space taken by comments. This is over 4 times the size of listing of GPSR, which is probably more densely coded. GPS-2-2 used 6.9 K words, about a third the size of GPSR, including the space used for a single task specification. GPSR can also be compared to a current Lisp version of GPS, called Mini-GPS. Mini-GPS differs from GPSR in having only Form operators, in lacking object and goal recognition processes, and in being restricted (somewhat specialized) to performing a symbolic manipulation task similar to the earlier GPS logic tasks. It has 66 Lisp functions and its program listing takes about 616 lines. The program listing for GPSR takes about twice as many lines, and the number of GPSR's Ps is about double the number of Mini-GPS functions, making allowances for Mini-GPS's functional limitations. But since most Lisp functions (this is somewhat speculative) test more than two conditions (i.e., alternatives in Lisp COND's), it must be the case that a P is expressing action at a higher level, since by this there is much more than one condition-action pair in Lisp corresponding to a P. This must be taken only as a preliminary line of thought, to be properly treated more fully in a context where a more satisfactory comparison can be made.

One feature that is very useful but that occasionally causes problems is the "parallelism" of P firings. This refers to the simultaneous firing of a P whose condition matches in more than one way. All of the possibilities are used, and consequences are followed up in unpredictable order. This makes processing sets of similar items appear as if the processing is happening to only one, as far as the static program is concerned. For instance, the Match-Diff submethod is expressed as a set of Ps that test the various special cases, but the control of where in a tree-structured object the match is located is left open to the arbitrariness of the firing parallelism. As with asynchronous processes in general, it is necessary to bring together (synchronize, or join) the various computational strands; in PSs this is done by a single P that uses the results of such a process but whose condition requires that no loose ends are lying around. This is necessary because, once control passes on, the loose ends get masked by new processing (pushed down in the stack, :SMPX) and do not rise to the top until some later point.

This parallelism is used in several processes in GPSR. Feasible assignments are generated by multiple firings of single task Ps (actually one task P for each class of desired assignment), and there is control, M40R, to make sure all feasible assignments are generated before a selection is made by M40 (that control is used only in case of multiple operators that give rise to more than one class of desired assignment). The Try-Apply submethod processes sets of feasible assignments in parallel, with the final selector for the result to which control is to pass, M45, acting as the synchronizer for the process. The desirability selection process constructs desirable assignments that may be sets of values for a particular variable; it generates assignments for different components of a difference in arbitrary order; and it may be working on assignments for more than one operator at once. M37, M37R, and M37S are the synchronizing Ps. The match also operates in parallel, as explained above, and also in the sense that it can do two or more matches at the same time. This is used in the MATCH:DIFF1 variant, called by the object filling process, in which several objects similar to an object may need to be matched, and in

● Implemented by J. Roehrich for instructional use in the CMU AI course.

which several objects may be being filed at once. The ordinary match is synchronized by using the stack (:SMPX) to hold off proceeding until all the match signals have been cleared out of the stack, letting the TRANSF2 signal come to the top; see Ps M20 and M23. The match used by the object filing is synchronized by F42 and the Ps that follow it. In general, Ps for erasure of data use multiple firings to advantage, e.g., the F20's, M37, and M44E. Finally, there are several places where complex selections are broken up into cascades of tests, with one P narrowing down a large set to a smaller set of candidates, and with the second P narrowing that set down to a unique selection. This is used in the Try-Old-Goals process (Ps E30-E32), in the selection of objects by the NEW:OBJ criterion (E35-E37), and in matching apply goals with various desired assignments, in the goal-recognition process (F8N-F8Z).

In most of the above examples, GPSR has extra mechanisms to synchronize scattered parallelisms, even though in ordinary processing it is unused, because the parallel firing paths have remained locked together. In other words, GPSR is written to produce the same behavior whether or not there is this parallelism, for the reason of making the code general enough to handle sets where ordinarily there are only single elements. Speaking abstractly two varieties of mechanism are used for this: first, using :SMPX to order the examination of two signals, where the first initiates processing and the second does the right thing afterwards - the second signal being held in the stack until the desired process has exhausted itself, at which point it becomes the most recent unexamined event, resulting in firing a P that renames outputs and initiates a process that uses those outputs; second, having the final Ps of a process test for any unused control signals for intermediate steps in the process, and if any are around, re-assert them so they become more active and precede anything done by the succeeding process, which uses the newly-developed results. Those places in GPSR where it was unnecessary to use such mechanisms because the parallel firings were guaranteed to stay together would have to be modified in one of those two ways, if that assumption were relaxed, and this would be only a minor inconvenience (no other known PS architecture has the feature). But if the program's static form required slight modifications, its dynamic performance would suffer more since the existing synchronizing mechanisms would actually be brought into use in most cases. All in all, parsimony favors retaining the multiple-firing feature, since there is no evidence that its negative aspect, too much action, is uncontrollable.

In addition to the control topics just discussed (namely, iteration over sets, synchronization of parallelisms, selection, combinatorial generation, and the match's scatter order), there are some other aspects, touched on elsewhere in this chapter, that can be summarized here. The executive's control context is maintained by a small number of data instances, telling it the status of the current goal and how to proceed if the goal should succeed or fail. Generally, local memory instances tell processes at all levels which subparts of processes (subsets of Ps) are responsible for the next decisions in the normal control flow. Either a process subpart knows or determines what is to come next, and asserts the appropriate data, or it simply asserts its output and the direction of control flow is determined by other instances that were stacked up (:SMPX) when the subpart was evoked. That is, control simply falls back to pending things to be done. The variant of the Match-Diff submethod that produces only a single difference is controlled (terminated) by having its control signals erased when enough of its output has been received. The overall organization of GPSR into executive + methods + processes + task-dependent operators is a conventional hierarchical organization, but achieved with an unstructured

set of Ps. The knowledge in the Ps, however, is rather modular, in the sense that Ps group naturally into sets according to the kinds of knowledge they represent. Ps in a module share much more with each other in terms of processing assumptions and data instances than Ps in separate modules do (see Section E.5). Perhaps we have traded modularity of knowledge for elaborate control structures.

GPSR is comfortably within an order of magnitude of reasonable run-time efficiency, either for human or computer. It solves the tasks exhibited in run times ranging from two minutes to about 45 minutes. This represents a range of 12 seconds per goal to about 40. By this seconds-per-goal figure alone, GPSR is about 3 to 10 times slower than GPS is estimated to be on the same computer (using Ernst and Newell's data, 17 seconds per goal on a 7090), and about 20 times slower than the Mini-GPS program. Most of the reason for this range is the increasing inefficiency in accessing Working Memory items as the number of items gets larger (for instance, all objects' representations are kept in Working Memory); this feature is tolerable at present because of the experimental first-approximation nature of Psnlst.[•] Times for the average Working Memory action range from 95 to 185 milliseconds, and for P firings from 335 to 605 milliseconds. The GPSR program uses 23.1 K 36-bit words (using the LISP encoding), and the task Ps use amounts from 3K (for MK) to 5.1 K (for MC). On the average this is about 10 K per 100 Ps. For Working Memory during the problem runs, from 6 K to about 20 K words are used. The only comparable figures available for GPS are that program and data used 20 K IPL symbolic locations, which would probably correspond to 20 K words on the computer used for GPSR. Mini-GPS's size is about 4.5 K. The only impact of the low run-time efficiency on implementation time was that only one or two test runs could be made per day. This was offset in part by being able to debug GPSR using only the printed trace and the trace of P firings; that is, debugging interactively was necessary only in the initial stages. The time inefficiency is actually only barely tolerable; that is, if it were somewhat worse, accomplishing anything would become almost impossible. Current expectations are that significant improvement is possible through radical changes in the implementation of Psnlst.

E.2. Trade-offs between Working Memory and Production Memory

In PSs in general, there are two ways of storing data: as items in the Working Memory (WM) or as data in RHSs of Ps (Production Memory, or PM) that can be evoked when needed. In psychological modelling, there is usually some theoretical constraint on which memory can (must) be used, usually determined by limits on WM capacity or WM item lifetime. But for AI purposes, and in particular for GPSR, the only strong consideration is programming convenience and efficiency. Nevertheless, for illustrative purposes PM is used for storing longer-term data in one instance, and it is possible on the basis of the experiments already completed to suggest other places in the program where this might be advantageous. The advantage of using PM is to reduce the size of WM, making matching more efficient by reducing combinatorial explosion, and removing data from the focus of attention so it doesn't interfere. There is a certain cost involved, though, in storing into and retrieving from PM. It is assumed that adding to the number of Ps does not increase the cost of finding a P that matches, since such Ps are usually keyed to signals that don't get used in the rest of the program.

[•] See Section G.2 for a discussion of improvements within the bounds of the PS language.

The illustrative use of PM is to store the components of loc-progs. These are used only occasionally, and their number for a task may get large (it is task-dependent). The desirability selection process uses them, so when it needs to have them, it emits a signal and the appropriate P responds by asserting the components. The components are then deleted on use. The component P is built at the same time as are the loc-prog recognition and application Ps, in the filing process.

A related potential use of this mechanism is in storing move-operator components, which are used in a way similar to loc-prog components. The revisions required to the desirability selection process (Section D.3) could use the PM to advantage, since erasing the components after use would then be possible, and their absence could be used as an indication that the process of constructing a desirable assignment was successful. Using their absence is somewhat less clumsy than using some new positive data items to signify that fact. This particular trade-off has emerged from experiments as useful, and thus is recommended, but in general such considerations are not easily foreseeable. Making this conversion is not anticipated to be difficult; one option is to simply force the task specification to bring about the switch, but the necessary Ps could also be built internally from presently-given data.

We have seen above, Section B.3, that there was a rich variety of decisions along this dimension for the canonization and recognition processes. For instance, the table of connections is expressed presently as an object in the WM, but could be expressed, in circumstances demanding more complex choices, as Ps. Another instance of a similar usage is expressing described objects as Ps in the MK task. There the match to an object is specified as a procedure that recognizes differences. We will discuss possible further modifications relating to this issue in Section G.2.

E.3. The ease of extending GPSR

A set of modifications were made to GPSR as the implementation developed. The ease with which this was done will illustrate the usefulness of GPSR for performing further GPS experiments. This subsection discusses the assignment ordering heuristic used for MC, the generalization of the transform method to allow it to be retried, the addition of the NEW:OBJ criterion for MC, the use of described objects, and the modification to the MATCH:DIFF submethod to produce a single difference rather than all differences and to perform more than one match at a time.

The assignment ordering heuristic in MC consists of preferring to have two people in the boat when moving it from left to right, and preferring only one in the boat on the return trip. To do this, two Ps, QF3 and QF4, were added to evaluate generated feasible assignments, and a change was made to M40 in the Try-Apply submethod to be sensitive to the evaluation. The two QF Ps were necessary because of the two different evaluations, one for left-right moves and the other for right-left moves. The change to M40 consisted of a nested sequence of conjuncts to cause selection to be based on the numerical value produced by the evaluation. It was necessary also to add an argument to the new-feasible-assignment signal (HASNEWFEAS), for the value (this affected about a dozen places in the program, but only the M40 location was sensitive to the addition). The two QF Ps are interposed between the generation of feasible assignments and their use

for testing applicability of an operator by renaming the output item (HASNEWFEAS) from the generators to serve as an input (HASNEWFEAS:ORD) to QF3 and QF4; these then had as their output the previous generator output item. To disallow the option, it is only necessary to change the name of the generators' outputs back to HASNEWFEAS. For tasks that have no assignment evaluation, the value argument is simply given as 0, amounting to a dummy placeholder.

The RETRY:TRANS option allows the Transform method to be retried by enabling it in the executive and by saving alternatives as they are generated. Ordinarily the executive stops backing up when it hits a transform goal, and evokes the Try-Old-Goals process; by adding an extra condition in E24 to test whether the option is on, and by adding E25 and E26 to apply the option and to fail on exhaustion of alternatives, the executive's action in this case was modified. The Transform method itself had to be modified to save alternative differences (from MATCH:DIFF) instead of erasing them. M24S does this; only differences whose difficulty is equal to that of the hardest difference are saved (as HASALT:DIFFR instances). (The Transform method makes an arbitrary choice from the set of equally-difficult differences.) The option is turned on by inserting the RETRY:TRANS signal at the beginning of a test run, manually.

The New-Obj option is implemented as a side case to the Try-Old-Goals selection. It selects an object that has not been the subject of a transform goal, creates a goal for that, and proceeds. There is a test for the option in E30 (the option is enabled by the SELECT:NEW:OBJ signal, inserted manually), and E35, E36, and E37 carry out the associated selection, goal construction, and cleanup operations. When no objects are available, Try-Old-Goals is done, by default.

Extending the Transform method to allow described objects as desired objects requires only one change in GPSR itself, and the addition of the requisite tests as part of task Ps (see the MK task P QK). M25 in the Transform method is necessary in order to detect a successful goal by noting that no differences are produced by MATCH:DIFF. Ordinarily this test is unnecessary because success can be detected by identical (canonized) object names. The described-object tests themselves respond to the MATCH:DIFF signal and produce output similar to that produced by the MATCH:DIFF submethod. The MATCH:DIFF submethod cannot do anything because a described object doesn't have any nodes for it to even get started with. The option is enabled simply by asserting ISDESCRIBED:OBJ for the desired object of the top goal (e.g., in Q1 of MK).

For use by the object-filing (canonizing) process, the MATCH:DIFF submethod needs only to produce a single difference, to distinguish two objects in the object network. The MATCH:DIF1 signal is set by the filing Ps, and the MATCH:DIFF submethod is evoked. That submethod works as it normally does, except at the final output stage, where there is a split according to whether MATCH:DIF1 is on. The split (an extra condition in K9) prevents the firing of K9, and K11 is used instead; K11 is the only added P for this change. Since the MATCH:DIFF submethod works in "parallel", usually not all of its results are produced at once; this can be used to advantage here because only one result is needed. The superfluous processing is prevented by erasing any remaining MATCH:DIFF data (Ps F21 through F28 do this). The added requirement that the MATCH:DIFF submethod be able to do two matches at the same time is achieved by adding two object arguments to MATCH:DIFF; this allows results from separate matches to be distinguished where previously they would not have been.

In summary, each of the five modifications discussed above was achieved very concisely, requiring at most the addition of three Ps and the modification of one existing P to allow a new branch in the execution path.

E.4. Mapping the GPS external representation onto the task Ps

In Section D.6, we have presented a comparison of features of the GPS external task language and GPSR task Ps. That can be summarized as follows. The mixed declarative-procedural aspect of PSs is used to fullest advantage when tests, moves, expressions, and variable constraints are expressed directly in task Ps. It is used less directly in the encodings of the DIFF:ORDER and TABLE:CONN objects: they are passive data structures, but they are accessed by task-specific loc-prog Ps that are built as required by GPSR. This passive encoding is only an expedient here, since nothing prohibits their being encoded as more general Ps that respond on request with the desired information. In a couple of cases it is necessary to represent things dually, once in Ps and once as Working Memory instances. This is due to the requirements of the desirability selection process: it must have information about specific operator effects (move-operator components) and about variable domains. An interesting idea developed in Section E.2 is that the information in Working Memory is more properly thought of as coming from the RHSs of Ps that are evoked when it's needed. Thus the duality of knowledge representation becomes that it is encoded in LHSs and RHSs of Ps. Ideally, of course, all of this would be supported by having a set of Ps to translate from an external language to the form usable by GPSR. My justification for not carrying this out is that such a sophisticated translation was not part of GPS either: its external language was artificial, and even though, perhaps, it wasn't as far away from natural language as the PS representation, the translation problems are similar.

E.5. The knowledge encoded in the GPSR executive

The approach to problem-solving that is embodied in GPS is to set up an executive whose expertise is evaluating progress and allocating effort among a set of methods for achieving various kinds of goals. The methods in turn produce results by making use of only task-specific knowledge. Thus there is a natural modularization of bodies of knowledge along the lines of this division into executive and methods and task-specific knowledge. In GPSR, the components of each body of knowledge are implemented as Ps, and each body is modular in that contact between bodies is rare compared to interactions within each. In this subsection we will consider briefly the nature of the bodies of knowledge, and then focus on how the knowledge in a part of the executive gets represented as Ps.

The body of knowledge in the executive deals with difficulty of goals and with goal-subgoal and antecedent-goal structure. It decides when to evoke a method to achieve a particular goal and it knows how to use the results of a method evocation to make further progress towards solution of the problem. The methods embody a set of general techniques that are applicable to a variety of specific tasks. The methods perform matching, evaluate differences resulting from the matching, select task operators according to appropriateness, keep track of alternative operators to try, and interface to the task

operators, specifying how the operators are to be applied and interpreting the results of application attempts. The knowledge that allows a specific task to be done deals with applying operators, extracting operator differences, generating feasible assignments, and initializing the task.

The executive is divided into several pieces: one evaluates new goals, one handles the success of goals, one handles failure, one checks for retryable supergoals and antecedents, and one selects old goals for retrying (the Try-Old-Goals process). We will focus on the evaluation piece, which is Ps E1-E8, corresponding to APs E0b, E0c, and E0d in Figure C.2. Figure E.1 gives slightly abstract Ps for E1-E8.

```

E1:  eval-goal & not repeated & not too-difficult -> select-method;
E2:  eval-goal & too-difficult & not repeated & isreduce-goal
      -> print("No Progress") & check-retry(supergoal);
E2R: eval-goal & too-difficult & repeated & isreduce-goal
      -> print("Repeated") & try-old-goals & disallow-retry;
E3:  eval-goal & too-difficult & hasantecedent-goal
      -> print("No Progress") & fail & methods-exhausted(antecedent-goal);
E4:  eval-goal & too-difficult & not hasantecedent-goal & isreduce-goal(supergoal)
      -> print("No Progress") & fail & methods-exhausted(supergoal);
E8:  eval-goal & repeated & not too-difficult
      -> print("Repeated") & try-old-goals & disallow-retry;

```

Figure E.1 Slightly abstract Ps for goal evaluation

Rather than take the Ps themselves as primitive units of knowledge, it is useful to state the knowledge more declaratively[•]. The following statement of the subset of the knowledge in the executive that is used in E1-E8 is intended to correspond to a more natural and immediate statement of knowledge contained in the bodies of knowledge sketched above, and to correspond to the form in which knowledge is first verbally formulated when something new arises as the result of experiments with GPSR.

- N1 a. Goal evaluation is the last thing that is done to a new goal before the executive selects a goal for further problem-solving effort;
 b. in the executive's selection, a new goal is preferred to old ones;
 c. a goal is evaluated as too difficult if it is more difficult than either its antecedent or supergoal.
 Used in: E1-E8; the "eval-goal" signal is implicitly a new "eval-goal"; how the evaluation is defined is used wherever "too-difficult" occurs.
- N2 If a goal is selected for further effort, a method should be selected to work on it.
 Used in: E1 (E22, E31).
- N3 A goal that is a repetition of a previous one should be abandoned permanently.
 Used in: E2R, E8.
 Interacts in: E1, E2.

• The approach here was first presented in Rychener, 1975.

N4 A reduce goal that is too difficult may be retried later, if no other reasons prohibit that retry; it should not fail, and its supergoal should be checked for the possibility of retrying.

Used in: E2.

Comment: the part about retrying the supergoal is probably wrong, from a GPS standpoint (it does describe GPSR's action); but this case does need to be distinguished from the ones below.

N5 When a repeated goal is abandoned, the Try-Old-Goals process should be evoked, if no other action is prescribed.

Used in: E2R, E8.

N6 A goal that is too difficult fails.

Used in: E3, E4. (Note that this overrides N5, since it is unqualified.)

N7 When a goal is too difficult, and when it has an antecedent or when it has a reduce supergoal, the antecedent or reduce supergoal should not be retried because any further tries would also be too difficult.

Used in: E3, E4.

N8 A reduce goal is derived as a subgoal of a transform goal or of an apply goal; it can never have an antecedent.

Used in: E3, E4; to remove the need for "not isreduce-goal". (This makes N4 and N7 mutually exclusive.)

Comment: Similar defining statements could be made for other goal types, but they aren't used in E1-E8.

Figure E.2 gives a picture of the mapping of knowledge statements to E Ps.

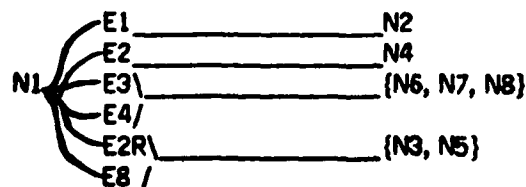


Figure E.2 The mapping between N's and E's

Several features are noteworthy. In E1 we see a typical effect of an interaction of three pieces of knowledge, N2, N3, and N6. E1 would start out as "eval-goal → select-method", based on N2; N3 and N6 each add a condition, to result in the E1 displayed. Similar interaction occurs in E2-E8 between how to handle goals that are repeated vs. how to treat the case where a goal is too difficult (non-progressive), both of which can be true of a new goal. The knowledge about how to treat goals that do not make progress (are more difficult than their supergoal or antecedent) arose out of experiments. For instance, N7 becomes necessary when one notices a goal that is retried several times, sprouting subgoals that are in each case too difficult. This knowledge takes advantage of the fact that subgoals are generated in order of non-decreasing difficulty. N4 also arose out of experimental need, as was discussed in connection with the MC task, Section D.5. Note that E2R, E3, and E4 do not exhaust all the cases of goals that a priori might be possible if

assigning difficulties is not taken into account. It was found by experience, however, that those three Ps do suffice, as a result of the way difficulties are assigned.

Knowing the history of how the executive developed from experiments and knowing by analysis the form of knowledge interactions in executive Ps leads us to conclude that further knowledge can be encoded in the same way as the examples above. That is, new knowledge will primarily result in new Ps, but is also likely to have interactions with existing knowledge, resulting in conditions added to existing Ps. This view is supported by the ways in which some extensions were made to GPSR, as we have seen in Section E.3.

F. GPS features of GPSR

F.1. Features of GPS that can be incorporated into other problem solvers

The organizational structure of GPS has been preserved in GPSR, and it is sufficiently useful to warrant emphasis here. The organization is based on the idea of an executive wandering around a goal-subgoal network structure, providing inputs to methods, evaluating results of methods, maintaining progress, and allocating effort (see Newell, 1962, for further discussion of this, and of alternatives to and history of this approach). This cleanly partitions expertise on various aspects of problem solving: the executive knows about goal tree structure, progress, and, at a general level, about specific methods; the methods are the repository of specific techniques, but they are interfaced to the executive in uniform fashion, they don't communicate with each other except through the executive, and they carry out relatively small and manageable pieces of the task (relative to everything required for a solution). This structure could presumably be expanded by adding new problem-solving methods, and the expansion would be supported by the existing structure. For GPSR, adding a method would require mainly extending the method-selection network, and the method would have to adopt or augment existing communication conventions; if new goal types were required, the goal-recognition and goal-sequencing processes in the executive would be affected.

Another attractive aspect of GPSR with respect to GPS mechanisms is the possibility that the task Ps can become arbitrary programs. In the examples done by GPSR, the program nature of the task information was only moderately exploited, but the general loose interface between the GPSR Ps and the task Ps might be used to advantage for more demanding tasks, or for tasks that deal with a much richer environment. As the task Ps grow, GPSR might become a minor, but important, subsegment of the overall problem-solving process.

In Section G.2 we will discuss some possible ways in which GPSR might lose its generality on a specific task, as it develops specialized Ps for performing some of its basic processes. This will include, for instance, building specialized Ps to recognize differences between objects and recognizing situations similar to previous ones where a successful sequence of operators might be directly applicable, thus formulating a plan spanning several goals. Even with such specialization, the executive-method organization would be retained as the indispensable goal-seeking and progress-maintaining core.

F.2. Problems with GPS from an implementation viewpoint

It was necessary in implementing GPSR to make interpolations at various places between what is described by Ernst and Newell and what is necessary to have a working program. This subsection will briefly discuss some of the problems encountered. In comparison to most AI programs, GPS was described in extensive detail, although it was also sufficiently complex to warrant that detail. Perhaps the parts of it that were unspecified relative to the PS implementation indicate as much about the level of the PS

language or about the PS organization of the GPS processes as they indicate about minor defects in the original description. That is, the small number of additional details that were essential to implementing GPS as a PS indicates the small gap between detailed informal description and a PS implementation. Some of the following however are clearly independent of implementation language.

Some details of lower-level processes were insufficient. One of the hardest to implement was the desirability selection process, by which desired (partial) assignments for operator applications are constructed. Connecting differences found by the GPSR match with specific operator information was difficult because it required decisions as to how much task information could be used without losing the generality of the process. The general solution sketched in Section D.3 was arrived at after experience with the requirements for several varied tasks.

Ernst and Newell didn't give enough details on the difference evaluation and on the scheme by which difficulties were assigned to goals as a result of that evaluation. It is evident from detailed study of GPS traces (in Ernst and Newell) that certain types of differences were ordered before others. For instance, in TH, reducing an UNDEF to a YES at some location is always preferred to reducing a YES to an UNDEF at another equally difficult (by DIFF:ORDER) location. This is probably most sensible, but it is an assumption about the problem-solving process that was not discussed. GPSR evaluates differences by assigning a numeric difficulty value according to heuristics discussed near the end of Section C.2. This value is then used as a difficulty value for goals. Such precision might have hidden dangers in general because difficulties are used to reject (sometimes only temporarily) goals from consideration, so that making too fine a distinction between differences could be putting too much precision at a place where some margin of error is appropriate (see, for instance, Section D.5). At present, however, it is not a serious problem for GPSR.

The executive in GPS was described in considerable detail, except that for the purposes of GPSR, the wrong details were described, and some essential details were omitted or fragmented in several locations. Part of the difficulty is that certain GPSR executive mechanisms were formulated in GPS as methods, for instance, Antecedent-Goal method and Try-Old-Goals method. The GPS description had too much implementation detail (in particular, dealing with the interpreter for the method language, which is unnecessary in GPSR due to the expressive power of the Ps themselves); that is, the implementation detail was too fine and went into issues that were irrelevant to the GPSR implementation. The GPS description lacked detail on how the executive managed successes and failures of goals (what was said for the executive contradicted what was said for the antecedent goal method, so there must have been some further essential mechanisms), on how it managed to maintain a multiple-supergoal structure (which it said arose as the result of repeated goals), and on the use of goal difficulties to evaluate progress. Unfortunately, the traces of GPS's behavior suppressed these executive aspects completely.

Some of the mechanisms in GPS were not adequately justified, so for the GPSR implementation there was a tendency to remove things that looked unnecessary. Subsequent experiments, however, proved their necessity, and important facts about GPS were brought out, but it would have helped to have had a priori reasons. In many

respects, this chapter suffers from similar defects, but its scope is too narrow to be adequate in that respect. It is hoped that further GPS research might be oriented to a fuller justification (see Section G.1). For instance, from simply considering GPS behavior traces, it is not evident that it is necessary to recognize repetitions of all three types of goals. One can plausibly reason that since reduce goals are derived from transform and apply goals, any repeated reduce goals would have to be derived from repeated goals, so that it is pointless to recognize reduce goals. This turns out in practice to be false: different goal contexts can in fact give rise to the same goals.[•] Further, one might reason that apply goals need not be recognized, in a similar way, but it turns out that it is possible to get into infinite loops of apply goals that don't include repeated transform goals; this occurs in unlikely-appearing sequences where apply goals are being retried.

One other unjustified mechanism in GPS is the Transform-Set method, which (seemingly arbitrarily) picks an object that has not previously been used as the subject of a transform goal, and constructs such a transform goal (this is the New-Obj criterion described above, near the beginning of Section C.2; the method in GPS has become part of the executive in GPSR). This method is "justified" by explaining that it works properly, since transforming an object derived from the initial object to get to the desired object is logically equivalent to the original problem. But it is not justified in the sense that, as we have seen in discussing the MC task (Section D.5), this method is essential to the success of GPS on that task: the rest of the methods without the Transform-Set method are insufficient to find a solution (thus, incomplete). This justification is important because the method seems quite irrelevant to means-ends analysis, since it amounts to arbitrary, undirected search, and since it doesn't use any means-ends principles. We will see below that it might be possible to improve the selection using means-ends mechanisms already in GPSR.

[•] Whether in this case they really are repeated is still a question, since they are repeated to attain different goals. This is especially critical if the repeated goal happens to be one that is easily attainable, as evidenced by previous success. Perhaps this is closely related to the multiple supergoal mechanism, since the success of a goal with many supergoals implies the success of all of them, even if they are not all known when the success occurs.

G. Topics for Further Research

G.1. GPS research topics

This subsection brings up topics related to developing GPSR, in four groups. The first group deals with problems with the theory of means-ends analysis and outlines the aspects of the problem-solving executive that might be subjected to variation in order to explore the space of GPS-like programs. The second group addresses possible augmentations in the basic set of techniques, in order to improve GPS within the means-ends framework. The third group considers changes in the way GPSR uses its basic techniques that might improve its performance. The fourth group consists of issues related to expanding the area of successful application of GPSR techniques. These topics are discussed here because implementing GPSR has raised new issues and has caused old issues to be addressed from a different viewpoint. GPSR is considered useful for further GPS research because of properties inherent in the use of PSs, which will be discussed in the next subsection.

The first group of topics addresses the theory of means-ends analysis. We have already discussed the apparent incompleteness of the basic means-ends analysis methods. We will discuss in the next paragraph some of the design decisions in the executive and methods of GPSR that might be varied to try to find a version with completeness. Regardless of whether completeness can be obtained experimentally, it would be useful to develop proofs of completeness or incompleteness and of task domain coverage: can the class of tasks be characterized abstractly, and can it be proven that GPS can solve that class (or a subclass of it). One minor aspect of GPS's search is that it seems to be aided by fortuitous orderings (of operators, of table of connections, etc.); we must ask what the ordering assumptions are and how they affect GPS's behavior. We must also examine ways of making the ordering more under the control of the means-ends heuristics (this is discussed further below).

A number of design decisions were made in the executive of GPSR. These might be varied in an attempt to improve the performance of GPSR or as an exploration of the design itself. GPSR retries old goals (Try-Old-Goals) when a goal is repeated (GPS was unclear on just what happened in this case); an alternative would be to retry the supergoal of the repeated goal. Retrying goals is propagated up the supergoal hierarchy, except through goals that have failed (in that case, Try-Old-Goals is done), until a transform goal is reached; perhaps Try-Old-Goals should be used sooner in this situation. GPSR does not retry transform goals, but evokes Try-Old-Goals; an alternative would be to go further up the supergoal hierarchy. GPSR is not opposed to retrying goals that were previously abandoned because they were more difficult than their supergoals; in fact this seems critical in at least one example (but the example was not tried without it); this may be due to too much precision in the difficulty scheme. GPSR does not have any multiple-supergoal mechanism; it is not clear how (or why) it was done in GPS, and no behavior trace exhibits its use. Since carrying over desired assignments led to completeness problems in GPSR, this should be investigated; GPS evidently had the same feature (in the MC and FS tasks) but it is not clear whether the impact was substantial, perhaps due to accidental orderings (GPSR and GPS do not exhibit exactly the same search behavior).

GPSR never retries apply goals as a result of Try-Old-Goals, even though doing so would result in new paths being tried (no examples exist that would seem to result in paths that would be real progress). The New-Obj selection in the executive selects the oldest object that fits the criterion, where some other order might be better, for instance the newest object, or the object closest to the goal. GPSR does things precisely in several places where perhaps more looseness would be appropriate: differences are evaluated precisely, difficulties are assigned using the difference evaluation, and desired assignments are made as precisely as possible, especially in MC where precise numerical values can be used. Goals in GPSR that have identical attributes are taken to be repeated, even when the surrounding goal-tree context is different and when the new goal has succeeded in the past attempt; in some examples it is clear that this might be inappropriate, and that some way of taking context and multiple supergoals into consideration is needed. Finally, the GPSR executive has parts that were separate methods in GPS. Revising this decision might be necessary if GPSR were to be applied to tasks beyond those done by GPS, but even then, the position taken by GPS might not be correct either, especially given some of the considerations above relating to alternative executive organizations.

The second group of topics addresses improvements of the basic elements that means-ends analysis has to work with. Some of these are raised also by Ernst and Newell. GPSR needs better differences, some of which were available in past versions of GPS, such as "size of object too large" or "this expression contains spurious C's". GPSR could perhaps record and make use of its history to extract, for instance, shortcuts that have been accidentally developed to attain previous goals. This is closely related to planning, one variety of which is to abstract move sequences from past behavior and generalize them to apply to other situations. The TH task is one example: the process of moving the two smallest disks is used several times in the solution of the four-disk problem. A second variety of planning in GPS has been demonstrated to be important, especially in the logic task (see Newell and Simon, 1972). This planning involves working with abstracted objects and operators, where some features are suppressed, so that such techniques might allow GPS to work with, for instance, a partially specified, vaguely described desired object.

GPS's behavior on certain graph-searching tasks (BK, WJ), where means-ends analysis provides almost no direction to the search, might be improved by adding more methods. GPS has difficulties with problems involving large objects (e.g., a chess board) and it has no satisfactory approach to handling data types such as sets with duplication, unordered sets, arrays, and so on. Some recent AI languages (the Planner-like languages, see Bobrow and Raphael, 1973) have in fact remedied some of GPS's problems with large objects and with diverse data types. We will discuss later in this subsection some ways in which GPS might develop into something like a problem-solving language.

The third group of topics addresses how GPS might be improved by making better use of techniques it already has. It might use its difference evaluation process to give each new object a distance-from-desired-object evaluation, thus perhaps allowing its search to be more directed by working first with objects that are closer to the desired one. This might, for instance, be used to adjust the difficulty of a goal, which at present uses the hardest difference as opposed to some measure based on all of the differences. This might relate to making GPS search look like Nilsson's A* algorithm (Nilsson, 1971). Also, as we have mentioned above (Section C.2), it might be possible to improve the object-filing process by making use of DIFF:ORDER to determine which attributes of objects are most likely to change and thus provide better discriminations.

The fourth group of topics deals with how to expand the area of application of GPS techniques. One way is to develop a process that takes natural language input, requiring GPS to build its own representation, table of connections, and difference ordering. One approach to this is put forth by Hayes and Simon (1973), although they do not carry the actual solution to completion, and they assume a more sophisticated GPS than really exists. Simon and Lea (1973) discuss the problem of how it is necessary to alternate between using problem-solving methods and processing the natural language input to get more information that might be useful, when the problem-solving process gets bogged down.

GPSR does not implement the GPS methods relating to form operators. These would require changes as follows. The match would need to be revised to work properly with unordered schemas. As it is, it assumes that there is only one way that sub-objects can be placed in correspondence, where, say, for algebraic expressions, several ways might have to be tried to find the best match, due to commutativity. The match would need immediate operators, such as assigning variables. The application of immediate operators creates variants on objects, which seems to affect object canonization in a way not presently accounted for. In general, objects would be less rigid in form, so that object-related processes would have to be generalized (loc-progs, canonization, table of connections, and difference ordering). Subgoals in problems with form operators would have to deal with subobjects, so that extra conventions to handle that would be required, e.g. in goal representation and in the executive. There are fundamental differences between form operators and move operators, so that basic processes like desirability selection would have to be altered. None of these changes requires major structural changes, except perhaps to the MATCH:DIFF submethod and to object canonization, so that the basic GPSR program structure would be sufficient.

Several processes in GPSR assume that generated sets are small enough that they can be generated in toto and processed, as opposed to generating single elements or small subsets and processing them before continuing the generation. GPS had two kinds of generators, Generate-and-Test for large sets and Select-Best-Members for small ones. In GPSR so far, no need for the large-set technique has been necessary, but other applications might require it. In particular, GPSR must generate feasible variable assignments for move operators, and it does so simply by computing the full set of combinations of variables and values. This has an effect on the Try-Apply submethod, which tries to apply all of the operator specifications resulting from those assignments, and then chooses the best for continuation (it can suspend the application testing if an operator applies successfully, but the remaining alternatives are still saved for future possible retrying). Whether generators need to be more conservative is task-dependent, and GPSR would need to be able to decide between the alternatives.

Finally, in exploring other tasks, it might be useful to open up the basic problem-solving processes in GPSR for task-specific adjustment. GPSR could develop into a problem-solving language, in the form of a powerful interactive manipulative system. This would not be dissimilar to the incremental simulation technique used by Woods and Makhoul (1974), in which a human user interacts with a program by filling in indefinite or unformalized sections of program action, gradually converting that action to actual program. This is feasible in the form of working with the actual Ps, whose flexibility is evident from Section E.3, where we discussed how readily the executive and methods were extended; from Section E.4, where we dealt with encoding task specifications; and from

Section E.5, where we dealt with encoding executive knowledge. The kinds of "advice" that would certainly improve GPSR's power in specific task domains are: planning as discussed above, of both types; improving the selectivity of the table of connections and of the desirability selection process; developing problem-specific differences and difference orderings; intervening at points where arbitrary selections are made; and adding new methods and expanding the method-selection network.

G.2. Production system research topics

PS research issues that are relevant to developing GPSR fall into four broad categories: efficiency, difficulties with implementation, design criteria for orientation to simulation of human problem solving, and expansion to more demanding task domains.

There are two ways to attack GPSR's inefficiency problems (Section E.1) at the external PS level (as opposed to changing the PS interpreter): reducing the number of P firings; and reducing the size of Working Memory, thereby making the match process faster by reducing the time to access memory elements. We will consider the various aspects of these two topics in the order of decreasing expected payoff. The primary approach to reducing the number of P firings is to find ways to collapse sequences of related firings into a single firing. In GPSR, this can be done by changing the mode of operation from "interpretive", where general Ps manipulate task information in the Working Memory, to "compiled", using Ps that are built to achieve the same effect but that are task-specific. The primary approach to reducing the size of Working Memory, which ranks second in terms of expected efficiency payoff, is to store much of the information in Ps when it is not needed for immediate processing, and to evoke the information from the Ps again when the need for it is recognized. An observed attribute of the P firing behavior is that a significant number of firings deal exclusively with erasure of items from Working Memory. As the third efficiency topic, thus, we will examine the possibilities for making erasure easier. Finally we consider the possibility of mixing Lisp code with the PS code to achieve efficiency in selected processes. These four efficiency topics will be discussed in order in the following paragraphs.

From the summary figures given at the end of the control flow trace (Appendix E) the F Ps, the M Ps, the K Ps, and the X Ps account for 60% to 70% of the P firings in GPSR. Since the M Ps are problem-solving methods, a high proportion there is inevitable, but the F's (filing), K's (matching), and X's (building external representations) should be less prominent if possible. Fortunately, the PS representations are amenable to collapsing sequences of related P firings, especially in the case of Ps that are interpreting task-specific structures. As we have seen in the PS representation of loc-progs, in contrast to the way the GPSR match works, it is possible to obtain a lot of action in a single P firing that is adapted to the task structure. So the basic idea in what is proposed now is to take further advantage of the fixed format and close similarity of task-specific objects. We should be able to do this wherever an interpretive node-by-node search of objects is done, or where components of a parameterized abstraction of node paths are examined singly.

An easy way to find pointers to places where some P-firing collapsing may be possible is the control flow trace, Appendix E. In that trace, P firings are grouped into

modules according to P initials, and the size of each group is given. Using these modules in this way assumes that they represent modularity of knowledge, and that collapsing firings across module boundaries is more difficult and has much less benefit because the aspects of the Working Memory that are touched on in different modules are almost non-overlapping. The following are suggested using a minimum of 5 P firings in one module. The match (K Ps) could be organized to recognize specific differences with single P firings, given the proper setup (similar to the loc-prog filing process); although some differences would be extracted much more easily, there is still the problem of determining what more has to be done in the match by the interpretive process. The operations evoked by task operators (T Ps) could be collapsed into a loc-prog access followed by the appropriate operation at the location. Similarly, with the proper variants of the loc-prog access mechanism, copying objects (C Ps), constructing desirable assignments (M32 ft.), and building the external representation of objects (X Ps) can be streamlined into fewer segments that do the same action as the interpretive node search. Perhaps the loc-prog filing process could process sets of links together also, after an initial determination of the size of loc-progs appropriate to a particular problem. Desirable assignment filing could be adapted to respond faster on known variable subsets composing the assignments. A few of the P firing sequences indicated as candidates for this discussion by the control flow summary trace are not considered here but are below with the erasure topic. Others are sequences of control that is not task-specific, so that a way is not yet seen to easily collapse them.

To get some idea of where it is necessary to reduce the size of Working Memory, we consider the state after the MC0 test run in Appendix F. Working Memory has about 1270 instances for that test (the number of instances at the end of tests ranges from somewhat more than that for MC1 down to about 350 for MK). Of those, 43% are used in instances dealing with the goal-subgoal structure, spread over 17 predicates, with a maximum per predicate of about 70 instances; 28% are used to represent objects, mostly in two predicates, LINKS and HASVAL, which are heavily loaded with 192 and 154 instances; and 13% are used in Try-Apply context that is saved to allow goals to be retried to explore alternate paths - of these, most (115) are in the ASSIGNS predicate, which holds operator variable assignments.

A more dynamic view of the memory state is given by the data-flow analysis summary at the end of Appendix E; that summary indicates a history of instances of each predicate in terms of how long the delay is between assertion and final use. Predicates whose instances have long delays are considered more global or long-term than others, so that these are candidates for being stored in Ps and evoked later when needed. By this, the goal representation, operator components, variable domains, desired assignments, and objects rank as most global; Try-Apply context, information on the created net Ps (desired assignment, object, and loc-prog), and loc-prog components are less global but can't be considered local.

Thus, according to both static and dynamic considerations, goals, objects, and assignments must be stored as RHSs of specific Ps, evoked whenever they're needed, and erased from Working Memory after use. This would reduce Working Memory to about 200 instances, with no predicates having a large number of instances, thus making the match more efficient by reducing access time for memory items. Of those 200, about 100 are instances of predicates unused except for debugging. This change in memory for goal

contexts would necessitate the use of an EPAM net for goal recognition, as opposed to the present specific Ps that match to all of the past goals in Working Memory. Changes would be necessary in the Try-Old-Goals process, with probably the necessity to record a goal's status in relation to Try-Old-Goals when it is erased from Working Memory. Also, the storage of objects in Ps makes representing TABLE:CONN and DIFF:ORDER as objects somewhat clumsy, so that a more suitable specialized representation would be used. With respect to efficiency, the other representations above are not important, but they will be discussed below in connection with further discussion of Working Memory reduction.

Erasure in GPSR is achieved in many cases by specific Ps that do only erasure, and these account for about 12% of all P firings on a typical test run. For instance, specific erasure must be done for intermediate results of the Match-Diff submethod (M22 ff.) and for stopping the match from generating more than one difference when filing objects (F20 ff.). These erasures are within P modules, so perhaps they are collapsable by the methods used above for other multi-P segments, but the problem is that what needs to be erased is somewhat variable. Three approaches deal more directly with erasure, as follows. More powerful erasure actions might be incorporated into the PS language, so that a single RHS action would accomplish the work of many P firings. The examples seen so far tend to be composed of instances that are readily described by simple patterns such as all instances of some set of predicates. Having Working Memory automatically fade over time would be another way of erasing unneeded elements, although there is need occasionally for explicit erasure, so this could not be adequate by itself. Finally, having the size of Working Memory fixed, with new elements replacing old ones according to a first-in-first-out discipline would have properties similar to the preceding.

Changing certain operations from the PS language to action (RHS) functions might improve efficiency in some limited areas. We have already discussed having more powerful erasure operations, which is an example of this hybridization. Another is more powerful P-building operations. It might be useful to have more power than just modifying current functions to be more convenient but using a similar number of P firings (such as are discussed below). A more powerful capability would involve, for instance, picking up a set of Working Memory instances, forming them into Ps by matching argument positions and generalizing constants. A third possibility for using better Lisp supporting functions is to make the interface between Working Memory and the external environment more automatic; for instance, this would replace the X Ps in GPSR by a single RHS action. Psnlst's limited macro capability only accesses Working Memory through variables bound by an ordinary PS match, but the more powerful function would access Working Memory directly.

We now turn to the topic of difficulties with the PS language from an implementation viewpoint: those features that made programming somewhat clumsy and debugging more difficult. In the next three paragraphs, we will discuss three difficulties as follows. The need for special Ps to do erasure has an impact on programming ease as well as on efficiency as discussed above. But there are also some important positive properties of erasure. The need for synchronizing "parallel" P firings as discussed in Section E.1 has undesirable properties of requiring Ps to be rather large and to include conditions about diverse kinds of knowledge. A better set of operations to add Ps and modify them will be presented with the aim of reducing the amount of list-processing and unnecessary manipulation in Ps.

There are 24 Ps in GPSR that do erasure exclusively, accounting for about 12% of P firings in a typical run, as mentioned above. In addition, erasure will become more prominent perhaps as more of Working Memory is converted to Ps as has also been discussed. Erasure is most clumsy in: removing intermediate data from Match-Diff; erasing unneeded match results; erasing objects that are duplicates of previous objects; terminating processes before they have fully run their course, for instance, Match-Diff1 in object filing and stopping the search for identical desired assignment when recognizing goals, both of which are instances of finding one member of a set that has some property; erasing unused choices in Try-Old-Goals; and erasing assignments for inapplicable task operators in Try-Apply. But erasure is also useful in some cases: in indicating that some data has been processed; in keeping track of context, when in a generation process; in stopping processes before completion or in general in interrupting processes; in indicating some property of a structure without making an addition to the structure - for instance, some goals are rejected from retrying in Try-Old-Goals by absence of a difficulty value; and in showing selection by erasing unselected candidates. Remedies for negative aspects of erasure have been discussed above.

Synchronization of parallel P firings, discussed in Section E.1, has two undesirable features from the programming viewpoint: it makes Ps rather large; and these large Ps use diverse knowledge and are not as localized and independent as Ps usually are, making assumptions about the related processes which are subject to change later. For instance, the F42's which do synchronizing of several matches potentially in filing objects, M40 which synchronizes feasible assignment generation for potentially many operators, and M45 which synchronizes collection of operator differences in Try-Apply. Also it can be clumsy to test and re-assert control signals in trying to finish up loose ends in a process that may have fired in parallel. One remedy is to remove the feature, which is discussed in Section E.1. A second is to add some control primitive that could be used to insure that all data asserted since some special signal or since some point in the P firing history has been examined; this might be more satisfactory than the present use of ad hoc data signals for the same function.

Present P-building operations are clumsy because they require an inordinate amount of list-processing in the RHSs of Ps, making the PS look more complex than it really is and reducing readability (see, e.g., F4, F34, F55). The actual operations used in GPSR are much less general than full list-processing, so that we can propose a set of better operations as follows: add P; extend LHS of P - e.g., as links are collected and tacked onto a basic P in building up a loc-prog recognition P; extend RHS of P; extend a nested conjunction inside an LHS - e.g., when a P has a nested conjunction that excludes conditions in another P and that other P is being extended; split a P by extending its LHS in two distinct ways, carrying over the same RHS in both; update an RHS conjunct, as when in splitting a P, a constant in one of the RHSs needs to be changed; maintain more information on LHS variables, and use it when extending an LHS (although perhaps this doesn't belong in the primitive set) - in GPSR some LHS extensions share one or two variables with the previous LHS but want most extension variables to be distinct from existing ones.

In GPSR, PSs have been used as a language with emphasis on convenience, expediency, and efficiency, rather than with a view towards accurately simulating human performance at the PS level. Considering a PS program as a human simulation imposes a number of constraints on the form of the program and its execution. P size in humans is

probably limited; that is, there might be limits on the number of conditions and actions in each P that would have to be taken into account in an implementation. Humans appear not to have the same kinds of arithmetic and list-processing primitives that are used in GPSR. In some cases, other primitives would be used, and in others, GPSR primitives would be achieved by sets of Ps. On the other hand, the power of the PS used in GPSR is probably much less powerful than the human control mechanism in some respects: the human might have a much more GPS-like match capability, with provision for dealing with partial matches and extraction of differences; and it probably has facilities for adding Ps and for collapsing sequences of P firings into fewer firings, perhaps as sketched above. But the most gross characteristics of GPSR that don't coincide with what is known about the human immediate processor are the size and persistence over time of Working Memory items.

We have discussed some of the ways GPSR could be changed above, in connection to making it more efficient, but the additional Working Memory issues to be discussed now are perhaps contrary to efficiency, and are based on the principle that Working Memory is small and short-term. All of the following issues are to be dealt with by storing information in RHSs of Ps, to be evoked on demand. Task-specific variable domains are presently asserted at the beginning of a problem and are used from Working Memory throughout a problem run. Context for the three networks of Ps that are built up by GPSR, consisting simply of pointers to the last Ps added, are kept as Working Memory items. Components for task operators are kept in Working Memory and used through the run by the desirability selection process; modification to have these stored as Ps would have the advantage that erasure after they had been used in constructing desirable assignments would be an indication of successful completion of that process. Finally, Try-Apply context, which is used when a goal is retried at some time later than its creation, could be stored as P RHSs, but other approaches might be more suited to the problem-solving methods. Which one of the following four approaches is most useful depends on the size of the set of alternatives that constitute the Try-Apply context, on how many times a generator of alternatives is restarted, and on how much of the generator's output is used each time it is started. First, a problem-solving method that can't do as complete a search of alternatives might be used. Second, generation might be more conservative - Ps could keep track of what has been generated so that the generator could resume. Third, the generator could generate the full set of alternatives once and an auxiliary P could store the part of the set that remains unused, with the RHS of the P being replaced as more is used or with a new P with a shorter RHS added and the old one masked by adding dummy LHS conditions (the assumption is that Ps can't be deleted). Fourth, the generator could create the full set of alternatives each time and auxiliary Ps could then consume parts of the set immediately to show which are undesirable due to past use.

Expansion of the task domain handled by GPSR might best be achieved by making GPSR more interactive, as discussed in the last few paragraphs of the preceding subsection. Here we discuss briefly how PSs have an impact on this goal. With respect to the incremental simulation technique, the PS step size and the GPS executive-method-task division are useful. The high level of PSs makes the expression of knowledge similar to the GPS method level, but of course the entire GPSR program is at that level and uniformly expressed. Thus, experiments with all GPS constructs are conducted in a language close to that method level. As we have seen, the openness of PSs makes extension easy (Section E.3), but PSs also seem promising with respect to making changes in natural language, and examining the knowledge content of Ps also in natural language. We saw an example of

natural language expression of knowledge in GPSR in Section E.5. Finally, maintaining a useful history of program operation is essential to useful interaction. For PSs, the history consists of the P firings, and if this is detailed enough, it can be used for backing up to undo a stream of behavior, for detecting common sequences so that shortcuts can be taken or planning in the senses discussed in the preceding subsection can be done, and for making analogies with similar previous behavior so that errors can be diagnosed, or so that methods can be generalized or extended to work in new situations. PSs can be used to detect conditions of program error, and simply to direct the process elsewhere using standard GPS executive processes, thus avoiding errorful areas of the search - this assumes that the methods are redundant enough that there is more than one method to getting past some obstacle. Having Ps detect and fill in partial matches by determining what is supposed to be there for a process to work (by analogy as suggested above) is also conceivable.

G.3. Production systems as a new level of problem-solving

From the work with GPSR, it is evident that PSs constitute a real advance in the nature of problem-solving languages and, by extrapolation, in organization of problem-solving programs. To see this consider three classes of problem solvers: GPS, theorem-provers, and PSs. The traits that should be emphasized for our comparison are as follows. For GPS, methods are powerful heuristics, allowing search through the set of problem states to be significantly pruned. GPS is limited to relatively small objects (problem states), and is limited in its ability to describe match differences. GPS is also limited in its ability to become tuned to particular tasks (but not GPSR, we maintain). The power of GPS to solve problems, and for problems to be expressed in language usable by GPS, seems to be good, but we have no rigorous proofs of task area coverage or language power. For theorem provers, strategies seem to be weak, since they are uniform procedures with search not so easily restricted. There are no limits on representational power as it pertains to objects or descriptions of differences, except that some solution to the frame problem must be used. Theorem provers are not obviously tunable to particular tasks. Expression of tasks for them is uniform and general. For PSs, we now can say that GPS's methods are usable. In addition, they can represent with the same power as theorem provers, but their expression is not limited to declarative (non-procedural) forms. PSs are tunable to particular tasks and open for modification, as sketched above, especially with respect to properties of GPSR's representations. In most respects, other new AI languages (see Bobrow and Raphael, 1973), for instance the Planner-like ones, are very much like PSs; the exceptions are the last two properties of PSs: there has not yet been a demonstration that they can become tuned to tasks, nor has their representation been demonstrated to be as much a mixture of declarative and procedural as have PS representations. Thus PSs are an advance in constructing problem solvers, in combining useful traits of GPS and theorem provers, and in addition having the traits of tunability to tasks and flexible representation.

GPSR's organization of executive + methods + task Ps is modular with respect to bodies of knowledge in each of the components, but in itself this is not a point of superiority over other languages. The essence of means-ends analysis is the particular combination of transform, reduce, and apply methods, so this is not likely to be a place to distinguish PSs from the others. Also, the processes that compose methods appear to be

modular and could be used in new combinations for new methods; other processes could likewise replace the present ones. But only the most general aspects of PSs seem to be helpful here, namely uniformity of expression, high level of expression, explicitness, and the global property of Working Memory. That is, we must look for PS advantages at more specific locations in the GPSR organization.

The GPSR organization is open for use as a driver or subroutine of some problem solver, as follows. There are three points where an interface to a problem solver could occur: the place between the external evoker and the GPSR executive; the place where the executive evokes the methods; and the place where methods evoke task Ps. For the first, GPSR is open because action is in small increments; this allows an outer process to interrupt at each executive cycle. The executive has very little control context: just sequencing commands from specific methods (that specify what is to be done on success) and the information stored in the goal network. Also the executive requires little input: just a few goal properties. The executive is independent of task representation, working at a rather distant level from the task; this may or may not be a feature that distinguishes the PS implementation from others. For the second interface, the only advantages from PSs are the uniformity of expression of all parts of the program and the high level of PSs as a language. For the third interface, having the task expressed as Ps is quite an advantage, apparently. Other languages distinguish too much between data and procedures to allow such flexibility.

H. References

- Bobrow, D. G. and Raphael, B. R., 1973. "New programming languages for AI research", Tutorial paper for Third International Joint Conference on Artificial Intelligence.
- Ernst, G. and Newell, A., 1969. *GPS: A Case Study in Generality and Problem Solving*, New York, NY: Academic Press.
- Feigenbaum, E. A., 1963. "The simulation of verbal learning behavior", in Feigenbaum, E. A. and Feldman, J., Eds., *Computers and Thought*, pp. 297-309. New York, NY: McGraw-Hill.
- Hayes, J. R. and Simon, H. A., 1973. "Understanding written problem instructions", in Gregg, L., Ed., *Knowledge and Cognition*, pp. 167-200. Potomac, Md: Lawrence Erlbaum Associates.
- Newell, A., 1962. "Some problems of basic organization in problem-solving programs", in Yovits, M. C., Jacobi, G. T., and Goldstein, G. D., Eds., *Self-Organizing Systems*, pp. 393-425. Washington, DC: Spartan Books.
- Newell, A., 1963. "A guide to the general problem-solver program GPS-2-2", Memorandum RM-3337-PR. Santa Monica, CA: The Rand Corporation.
- Newell, A. and Simon, H. A., 1963. "GPS, a program that simulates human thought", in Feigenbaum, E. A. and Feldman, J., Eds., *Computers and Thought*, pp. 279-293. New York, NY: McGraw-Hill.
- Newell, A. and Simon, H. A., 1972. *Human Problem Solving*, Englewood Cliffs, NJ: Prentice-Hall. Chapters 8-10.
- Nilsson, N. J., 1971. *Problem-Solving Methods in Artificial Intelligence*, New York, NY: McGraw-Hill.
- Rychener, M. D., 1975. "The Studnt production system: A study of encoding knowledge in production systems", Pittsburgh, Pa: Carnegie-Mellon University Department of Computer Science.
- Simon, H. A. and Lea, G., 1973. "Problem solving and rule induction: A unified view", in Gregg, L., Ed., *Knowledge and Cognition*, pp. 105-127. Potomac, Md: Lawrence Erlbaum Associates.
- Woods, W. A. and Makhoul, J., 1974. "Mechanical inference problems in continuous speech understanding", *Artificial Intelligence*, Vol. 5: 1, pp. 73-91.

GPSR

GPSR APPENDICES

Appendix A. PROGRAM LISTING FOR OPSR

```

BEGIN      1 OPS REVISED, IN PSMA ST. MAIN PRODUCTIONS 1

EXPR OPSR: BEGIN      1 NONFLUENT (HASVAL, LINES, ASSIGNS), DOND (OPBC)
      REQUIRE (OPN, GPSM, GPSL)

      1 GROUPS OF PRODUCTIONS: F M R T C D V X Q (AST, SPECIFIC)
      1N- LA- LC- (CREATED (PMT) OR- (ORJ NET) DA- (DA ME) 1

      1 PSMA CROS AND AUXILIARY FUNCTIONS:
      EXPLORE (X) FORMS A LIST OF THE CHARACTERS IN THE ATOM X OF READ 1ST
      LEXLE (A) TESTS IF A IS LEXICALLY LESS OR EQUAL TO B
      LEXLT (A) TESTS IF A IS LEXICALLY LESS THAN B
      NCONC (A) APPENDS THE TWO LISTS A AND B TOGETHER, DESTRUCTIVELY
      OBJECT (NAME, (A (B C D E) F (G H))) --
      HASSTOP (NAME, (N1) & LINES (AN1 N2) & LINES (RN1 N3)
      & HASVAL (N1) & LINES (DN1 N4) & HASVAL (N4, 1)
      & LINES (N1 N5) & LINES (CN1 N6) & HASVAL (N6, 1)
      1 NO HASVAL IF VALUE WOULD BE NIL 1
      READ 1ST (X) MAKES THE LIST OF CHARACTERS X INTO AN ATOM: C, EXPLORE
      SUBST (D, X) MAKES A COMPLETE COPY OF X COPYING ALL LIST LEVELS
      TRACE PRINTG PRINTS A TRACE LINE FOR A GOAL, ACCORDING TO 5 ARGUMENTS
      TRACE PRINTM PRINTS THE MESSAGE THAT IS ITS FIRST ARGUMENT
      XMERGE MERGES ALPHABETICALLY A PAIR INTO A LIST OF PAIRS

      1 EXECUTIVE 1

E0: "PRON IN" = GPSR IN (M) & EXISTSD (M) & ISLAMEY (M)
      & HASSTOP (M) & TRACE (IND) & LAST (PMT) (GPSL)
      & LAST ONE (GPSR) & LAST ONE (GPSL)
      & HASTRACE LEVEL (TOP) & NEGATE (1)

E1: "GOAL EVAL OK" = EVAL GOAL (GV) & NOT (EXISTSG (G)) & ISSAME GOAL (GG)
      & NOT (EXISTSV (V)) & HASDIFF (CGV)
      & SATISF (S2V1 V2 V2 %GREAT V1)
      1 SELECT METHOD (G) & NEGATE (1)
E2: "GOAL EVAL -" = EVAL GOAL (GV) & NOT RECOG GOAL (G)
      & HASDIFF (CGV) & SATISF (S2V1 V2 V2 %GREAT V1)
      & ISREDUCE GOAL (G) & NOT (EXISTSG (G)) & ISSAME GOAL (GG)
      & HASTRACE LEVEL (G) & HASSUPER GOAL (GG)
      1 TRACE (TRACE PRINTM) ("NO PROGRESS, G TAILED X")
      & CHECK RE TRY (G) & NEGATE (1)
      1 IF REDUCE WANT TO ALLOW RETRY LATER, SO FAIL IS RAD HERE,
      CHECK RE TRY ON SUPER IS OK 1
E2R: "GOAL EVAL R" = EVAL GOAL (GV) & NOT RECOG GOAL (G) & HASDIFF (CGV)
      & SATISF (S2V1 V2 V2 %GREAT V1) & ISREDUCE GOAL (G)
      & ISSAME GOAL (GG) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("REPEATED GOAL, GG 1 X")
      & TRY OLD GOALS (G) & NEGATE (1, 1)
E3: "GOAL EVAL A" = EVAL GOAL (GV) & NOT RECOG GOAL (G)
      & HASDIFF (CGV) & SATISF (S2V1 V2 V2 %GREAT V1)
      & HASANTEC (G) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("NO PROGRESS, GG TAILED X")
      & FAIL (G) & METHODS (X) (G) & NEGATE (1)
E3: "GOAL EVAL S" = EVAL GOAL (GV) & NOT RECOG GOAL (G) & HASDIFF (CGV)
      & SATISF (S2V1 V2 V2 %GREAT V1)
      & NOT (EXISTSG (G)) & HASANTEC (G) & HASTRACE LEVEL (G)
      & HASUPER GOAL (GG) & ISREDUCE GOAL (G)
      1 TRACE (TRACE PRINTM) ("NO PROGRESS, GG TAILED X")
      & FAIL (G) & METHODS (X) (G) & NEGATE (1)
      1 E3 AND E4 FAIL GOALS THAT MIGHT IN SOME FUTURE VERSION BE RETRIED,
      SO FAIL MUST NOT DESTROY THAT POSSIBILITY 1
      1 E4 IS CASE OF NO ANTEC, SO IT HAS OPDIFR AND IS NOT
      RETRIABLE ANYWAY 1

E4: "SAME REP" = EVAL GOAL (GV) & ISSAME GOAL (GG) & HASTRACE LEVEL (G)
      & HASDIFF (CGV) & NOT SATISF (S2V1 V2 V2 %GREAT V1)
      1 TRACE (TRACE PRINTM) ("REPEATED GOAL, GG 1 X")
      & TRY OLD GOALS (G) & NEGATE (1, 1) 1 1 OUT SO NO RETRY 1

E10: "SUC TRANS" = SUCCEED (G) & NEXT GOAL TRANS (G) & HASDIFF (CGV)
      & HASUPER GOAL (G) & HASTRACE LEVEL (G)
      1 EXISTSG (G) & TRACE (TRACE PRINTM) ("SUCCEEDS X")
      & FILE GOAL (G) & EVAL GOAL (GV) & ISTRANSFORM GOAL (G)
      & HASACTUAL ORJ (G) & HASDEST (DOR) (G) & HASANTEC (GG)
      & HASUPER GOAL (GG) & HASDIFF (CGV) & SUCCEED (G)
      & NEGATE (1)
E11: "SUC APPL" = SUCCEED (G) & NEXT GOAL APPL (G) & OPDA V
      & HASUPER GOAL (G) & HASTRACE LEVEL (G)
      1 EXISTSG (G) & TRACE (TRACE PRINTM) ("SUCCEEDS X")

```

```

      & FILE GOAL (G) & EVAL GOAL (GV) & ISAPPLY GOAL (G)
      & HASACTUAL ORJ (G) & HASDIFF (CGV) & HASDEST (DOR) (G)
      & HASUPER GOAL (GG) & HASOP (G) & HASANTEC (GG)
      & SUCCEED (G) & NEGATE (1)
E12: "SUC SUP" = SUCCEED (G) & NOT (EXISTSG (G)) & NEXT GOAL TRANS (G)
      & NOT (EXISTSTOP DA V) & NEXT GOAL APPL (G) & OPDA V
      & HASUPER GOAL (GG) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("SUCCEEDS X")
      & SUCCEED (G) & SUCCEED (G) & NEGATE (1)

E20: "FAIL ANTEC" = FAIL (G) & HASANTEC (GG) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("G TAILED X")
      & CHECK RE TRY (G) & FAIL (G) & NEGATE (1)
      & FAIL MUST NOT DESTROY CHANCE OF SUCCESS DURING THRU FROM
      SOME RETRIED SINGOAL 1
E21: "FAIL ANTEC" = FAIL (G) & NOT (EXISTSG (G)) & HASANTEC (GG)
      & HASUPER GOAL (GG) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("G TAILED X")
      & CHECK RE TRY (G) & FAIL (G) & NEGATE (1)
E22: "CHECK RE TRY" = CHECK RE TRY (G) & NOT METHODS (X) (G)
      & NOT ISTRANSFORM GOAL (G) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("RETRYING G X") & SELECT METHOD (G)
      & RETRY (G) & NEGATE (1)
E23: "CHECK RE TRY X" = CHECK RE TRY (G) & METHODS (X) (G) & NOT FAILED (G)
      & FAIL (G) & NEGATE (1)
E23R: "FAIL REP" = CHECK RE TRY (G) & FAILED (G) & TRACE (IND)
      1 TRACE (TRACE PRINTM) ("REPEATED FAIL G X")
      & TRY OLD GOALS (G) & NEGATE (1)
E24: "CHECK RE TRY G" = CHECK RE TRY (G) & ISTRANSFORM GOAL (G)
      & NOT (EXISTSG (G)) & RETRY TRANS (G) & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("TRANSFORM RETRY REJECT G X")
      & TRY OLD GOALS (G) & NEGATE (1)
E25: "RETRY TRANS" = CHECK RE TRY (G) & ISTRANSFORM GOAL (G) & RETRY TRANS (G)
      & HASAL TO DIFF (G, V1 V2) & HASTRACE LEVEL (G)
      1 TRANS (G) & MATCHVAL (V1 V2)
      & TRACE (TRACE PRINTM) ("RETRYING TRANSFORM G X")
      & NEGATE (1)
E26: "RETRY TRANS" = CHECK RE TRY (G) & ISTRANSFORM GOAL (G) & RETRY TRANS (G)
      & NOT (EXISTSG, V1 V2) & HASAL TO DIFF (G, V1 V2)
      & HASTRACE LEVEL (G)
      1 TRACE (TRACE PRINTM) ("TRANSFORM EXHAUSTED G X")
      & TRY OLD GOALS (G) & NEGATE (1)

E30: "TRY OLD GOALS" = TRY OLD GOALS (G) & ISREDUCE GOAL (G)
      & NOT (EXISTSG (X1 X2 X3)) & SELECT WORK (X1) 1 SEE E35 1
      & HASACTUAL ORJ (X2 X3) & NOT ISTRANSFORM GOAL (X2)
      & NOT (EXISTSG (X4)) & HASACTUAL ORJ (X4 X3)
      & ISTRANSFORM GOAL (X4)
      & NOT METHODS (X) (G) & HASDIFF (CGV)
      1 ASSUMES REPEATED GOALS HAVE DIFFICULTY REMOVED 1
      & NOT (EXISTSG (G2 V2)) & ISREDUCE GOAL (G2) & NOT METHODS (X) (G2)
      & HASDIFF (CG2 V2) & SATISF (S2V1 V2 V2 %LESS V1)
      & HASUPER GOAL (G1 G4)
      & NOT (EXISTSG (G3)) & ISAPPLY GOAL (G4) & ISREDUCE GOAL (G2)
      & VNE (G2 G1) & HASDIFF (CG2 V1) & NOT METHODS (X) (G2)
      & HASUPER GOAL (G2 G3) & ISTRANSFORM GOAL (G3)
      1 CHOOSE OLD GOAL (G1) & NEGATE (1) 1 1 FIFES MULTIPLE 1
      1 IF EVER DECIDE TO RETRY APPL'S, RE-CONSIDER E3 EFFECTS 1
E31: "CHOOSE OLD" = CHOOSE OLD GOAL (G1)
      & NOT (EXISTSG (G2)) & CHOOSE OLD GOAL (G2)
      & SATISF (S2V1 G2 G2 LEXLT G1)
      & HASTRACE LEVEL (G1) & TRACE (IND)
      1 ERASE CHOICE (G1) & TRACE (TRACE PRINTM) ("G 1 X")
      & TRACE (TRACE PRINTM) ("RETRYING OLD G 1 X")
      & SELECT METHOD (G1) & RETRY (G1) & NEGATE (1) & TRACE (IND)
E32: "ERASE CH" = ERASE CHOICE (G1) & CHOOSE OLD GOAL (G) & NEGATE (ALL)

E35: "NEW OBJ CRIT" = TRY OLD GOALS (G) & SELECT WORK (X)
      & HASACTUAL ORJ (G) & NOT ISTRANSFORM GOAL (G)
      & NOT (EXISTSG (G2)) & HASACTUAL ORJ (G2)
      & ISTRANSFORM GOAL (G2)
      1 CHOOSE OLD (G) & NEGATE (1)
E36: "CHOOSE ORJ" = CHOOSE OLD (G)
      & NOT (EXISTSG (G2)) & CHOOSE OLD (G2)
      & SATISF (S2V1 G2 G2 LEXLT G1)
      & TRACE (IND) & HASUPER GOAL (G1 G2) & SATISF (S2V1 G2 G2 TOP)
      & HASDEST (DOR) (G)
      & NOT (EXISTSG (G3)) & HASUPER GOAL (G3 G2) & UNQUINNESS 1
      & SATISF (S2V1 G3 G3 LEXLT G1)
      1 EXISTSG (G) & ERASE CHOICE (G1) & TRACE (TRACE PRINTM) ("G 1 X")
      & TRACE (TRACE PRINTM) ("SELECT BY NEW ORJ, D 1 X") 1 1 SKIPS 1
      & FILE GOAL (G) & EVAL GOAL (GV) & ISTRANSFORM GOAL (G)
      & HASACTUAL ORJ (G) & HASDEST (DOR) (G)

```

```

      @ HASHPROGGOAL(G2) @ NEGATE(3) @ TRACT(IND1)
E57: "ERASE CH" = ERASE CHOICE(SOCH) @ CHOOSE(CHOORL) @ NEGATE(ALL)

```

```

% THESE WERE USED FOR RECOGNITION:

```

```

E60: "RETRYAN F GOALS" = GENCANDOT(X) @ ISPRODUCE GOAL(G)
      @ NOT METHODSE(XMG) @ HASDIFF(IGV)
      @ NOT(EXIST(SO2 V2) @ ISPRODUCE GOAL(G2) @ NOT METHODSE(XMG2)
      @ HASDIFF(IG2 V2) @ SATISFIES2(V2 V2 "LESS V))
      @ ISCANDOT(G) @ NEGATE(1)
E61: "OTHER RETRYAN F" = GENCANDOT(X) @ ISPRODUCE GOAL(G)
      @ NOT METHODSE(XMG) @ HASDIFF(IGV)
      @ ISPRODUCE GOAL(G2) @ NOT METHODSE(XMG2)
      @ HASDIFF(IG2 V2) @ SATISFIES2(V2 V2 "LESS V)
      @ ISCANDOT(G) @ NEGATE(1)

```

```

END)

```

```

EXPR GPST(1): BEGIN      % FILING %      % PAGE 2 %

```

```

F1: "FILE LOC PROG" = FILE LOC PROG(X) @ HASHPROG(X) M      % LOC-PROGS %
      @ NOT(EXIST(SO2 V2) @ HASHPROG(X) M2)
      @ SATISFIES2(V2 V2 "LESS V))
      @ HASHPROG(X) M @ EXTEND(IND1) @ NEGATE(1)

```

```

% TARGETS FOR LOC-PROG MET
% A-3: "SAMPLE APPLY" = APPLY LOC PROG(G) @ SATISFIES2(L EQ "P-3")
      @ HASHPROG(X) M @ HASHPROG(X) M2
      @ SATISFIES2(L EQ "L INK") @ HASHPROG(X) M2
      @ LOC PROG(SO2 V2) @ NEGATE(1)

```

```

% N-3: "SAMPLE RECOG" = HASHPROG(X) M @ SATISFIES2(L EQ "L INK")
      @ SATISFIES2(M INI EQ)
      @ HASHPROG(X) M @ HASHPROG(X) M2 @ NEGATE(ALL)
      @ NOT(EXTEND(IND1))

```

```

% C-3: "SAMPLE COMPON" = GET LCOMPON(L) @ SATISFIES2(L EQ "P-3")
      @ HASHPROG(X) M @ HASHPROG(X) M2 @ NEGATE(1)

```

```

F2: "EXTEND LP MET" = EXTEND LP MET(D) @ EXTEND LP MET(D) @ NEGATE(1)

```

```

F3: "ST LP MET COL" = EXTEND LP MET(D) @ HASHPROG(X) M
      @ NOT(EXIST(SO2 V2) @ HASHPROG(X) M2)
      @ SATISFIES2(V2 V2 "LESS V))
      @ EXTEND LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

% HASHPROG(X) M @ HASHPROG(X) M2
% SATISFIES2(V2 V2 "LESS V))
% EXTEND LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

% HASHPROG(X) M @ HASHPROG(X) M2
% SATISFIES2(V2 V2 "LESS V))
% EXTEND LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

F4: "COL LP MET" = COL LP MET(D) @ HASHPROG(X) M
      @ NOT(EXIST(SO2 V2) @ HASHPROG(X) M2)
      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      @ COL LP MET(D) @ COL LP MET(D) @ NEGATE(1)

```

```

      @ HASHPROG(X) M @ HASHPROG(X) M2

```

```

      @ SATISFIES2(V2 V2 "LESS V))

```

```

      & ADDPROXONP(OR(X,TESTONE(X),HASTOPONE(X),X2),
        (ISSAME(X),X1,X1))
      & HASTOPONE(X2) & LASTONE(100)
      & NEGATE(1);
F16: "SAME SET" = TESTONE(OR(X,ISSAME(X),X2))
  >> TESTONE(OR(X,ISSAME(X),X2)) & NEGATE(ALL); & ACCESS IF NOT SAME'S
F16: "SAME OR" = TESTONE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X)) & ISSAME(X,OR(X))
  >> MATCH(OR(X,ISSAME(X),X2)) & MATCHES(X,OR(X)) & MATCH(OR(X))
  & NEGATE(ALL);
F17: "SAME EQ" = TESTONE(OR(X,ISSAME(X),X2)) & ISSAME(X,OR(X))
  >> MATCH(OR(X,ISSAME(X),X2)) & MATCHES(X,OR(X)) & MATCH(OR(X))
  & NEGATE(1);
F18: "LAST MLT" = LASTONE(100) & LASTONE(100) & SATISF(OR(X,ISSAME(X),X2))
  >> LASTONE(100) & NEGATE(1);
F20: "OBJ DIFF" = MATCHES(X,OR(X)) & MATCHES(X,OR(X))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  & SATISF(OR(X,ISSAME(X),X2)) & MATCH(OR(X))
  >> ERASE(OR(X,ISSAME(X),X2)) & ERASE(OR(X,ISSAME(X),X2))
  & EXTONE(OR(X,ISSAME(X),X2)) & NEGATE(1);
  & CASE OF NO DIFFER FORM - F60
F21: "ERS MD1" = ERASE(OR(X,ISSAME(X),X2)) & MATCH(OR(X,ISSAME(X),X2))
  >> MATCH(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F22: "ERS MR1" = ERASE(OR(X,ISSAME(X),X2)) & MATCHES(X,OR(X))
  >> ERASE(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F23: "ERS MR1" = ERASE(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  >> NEGATE(1);
F24: "ERS ML1" = ERASE(OR(X,ISSAME(X),X2)) & LOCEXTN(OR(X,ISSAME(X),X2))
  >> ERASE(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F25: "ERS ML1" = ERASE(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & LOCEXTN(OR(X,ISSAME(X),X2))
  >> NEGATE(1);
F26: "ERS MN1" = ERASE(OR(X,ISSAME(X),X2)) & MATCH(OR(X,ISSAME(X),X2))
  >> NEGATE(ALL);
F27: "ERS MN1" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(1);
F28: "ERS MD1 SIG" = MATCH(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & MATCH(OR(X,ISSAME(X),X2))
  >> NEGATE(1);
F30: "EXT ON ST" = EXTONE(OR(X,ISSAME(X),X2)) & HASTOPONE(X)
  & HASTOPONE(X)
  & NOT(EXISTS(X,OR(X))) & HASTOPONE(X)
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  >> COLONE(OR(X,ISSAME(X),X2)) & SUBST(OR(X,ISSAME(X),X2))
  << "LINES: T,X7 READIST(X CONS EXPLODE(N-1))",
    "SATISF(OR(X,ISSAME(X),X2))",
    "V1,V2,D1,07)
  & NEGATE(ALL);
F32: "EXT ON" = COLONE(OR(X,ISSAME(X),X2)) & HASTOPONE(X)
  & NOT(EXISTS(X,OR(X))) & HASTOPONE(X)
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  >> COLONE(OR(X,ISSAME(X),X2)) & SUBST(OR(X,ISSAME(X),X2))
  << "LINES: T,X7 READIST(X CONS EXPLODE(N-1))",
    "READIST(X CONS EXPLODE(N-1))",
    "SATISF(OR(X,ISSAME(X),X2))",
    "V1,V2,D1,07)
  & NEGATE(ALL);
F34: "SPLIT ON P" = COLONE(OR(X,ISSAME(X),X2)) & NOT(ISOAMN(OR(X)))
  & NOT(EXISTS(X,OR(X))) & HASTOPONE(X)
  & HASTOPONE(X) & LASTONE(100)
  & NOT(EXISTS(X,OR(X))) & LASTONE(100)
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  & HASTOPONE(X) & X RECOMES A COPY OF L IN RMS
  >> EXISTS(OR(X,ISSAME(X),X2)) & SPLIT(OR(X,ISSAME(X),X2))
  << "LINES: T,X7 READIST(X CONS EXPLODE(N-1))",
    "READIST(X CONS EXPLODE(N-1))",
    "SATISF(OR(X,ISSAME(X),X2))",
    "V1,V2,D1,07)
  & NEGATE(ALL);
F36: "SPLIT ON P" = COLONE(OR(X,ISSAME(X),X2)) & NOT(ISOAMN(OR(X)))
  & NOT(EXISTS(X,OR(X))) & HASTOPONE(X)
  & HASTOPONE(X) & LASTONE(100)
  & NOT(EXISTS(X,OR(X))) & LASTONE(100)
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  & HASTOPONE(X) & X RECOMES A COPY OF L IN RMS
  >> EXISTS(OR(X,ISSAME(X),X2)) & SPLIT(OR(X,ISSAME(X),X2))
  << "LINES: T,X7 READIST(X CONS EXPLODE(N-1))",
    "READIST(X CONS EXPLODE(N-1))",
    "SATISF(OR(X,ISSAME(X),X2))",
    "V1,V2,D1,07)
  & NEGATE(ALL);
F38: "SPLIT ON P" = COLONE(OR(X,ISSAME(X),X2)) & NOT(ISOAMN(OR(X)))
  & NOT(EXISTS(X,OR(X))) & HASTOPONE(X)
  & HASTOPONE(X) & LASTONE(100)
  & NOT(EXISTS(X,OR(X))) & LASTONE(100)
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  & HASTOPONE(X) & X RECOMES A COPY OF L IN RMS
  >> EXISTS(OR(X,ISSAME(X),X2)) & SPLIT(OR(X,ISSAME(X),X2))
  << "LINES: T,X7 READIST(X CONS EXPLODE(N-1))",
    "READIST(X CONS EXPLODE(N-1))",
    "SATISF(OR(X,ISSAME(X),X2))",
    "V1,V2,D1,07)
  & NEGATE(ALL);

```

```

      & DEPRHS(OR(X,ISSAME(X),X2),OR(X,ISSAME(X),X2))
      & HASTOPONE(X) & LASTONE(100)
      & NEGATE(ALL);
F36: "SPLIT ON P" = COLONE(OR(X,ISSAME(X),X2)) & NOT(ISOAMN(OR(X)))
  >> DEPRHS(OR(X,ISSAME(X),X2),OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  & NEGATE(1);
F38: "SPLIT ON P" = COLONE(OR(X,ISSAME(X),X2)) & NOT(ISOAMN(OR(X)))
  >> DEPRHS(OR(X,ISSAME(X),X2),OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  & NEGATE(1);
F40: "NO DIFF" = MATCHES(X,OR(X)) & MATCHES(X,OR(X))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  & HASTOPONE(X) & TRAC(OR(X))
  >> ERASE(OR(X,ISSAME(X),X2)) & ISSAME(X,OR(X))
  & TRAC(OR(X,ISSAME(X),X2)) & TRAC(OR(X,ISSAME(X),X2))
  & NEGATE(1);
  & IF MULTIPLE OBJ'S FILED GET TO F60 AND HIT SAME P, NEGATING
  THE HASHS IS BAD
  & NOTE THAT MULTIPLE OBJ'S FILED GETS TO A NEW OBJECT
  ARE ALL CARRIED THROUGH (F41,F42) EVEN IF THE NEW ONE
  MATCHES AN OLD ONE: THIS WILL SAVE EFFORT IF THAT OBJECT
  SHOULD COME ALONG AGAIN
F41: "CHK DONE" = ISSAME(X,OR(X)) & SUCCEED(OR(X))
  >> ONE(SUCCEED(OR(X))) & NEGATE(1);
F42: "ONE DONE" = ONE(SUCCEED(OR(X)))
  & NOT(EXISTS(X,OR(X))) & MATCH(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  & LOCEXTN(X,OR(X)) & NEGATE(1);
  & HASTOPONE ALWAYS ACCOMP BY LOCEXTN OR MATCHES
  & NOT(EXISTS(X,OR(X))) & TESTONE(X,OR(X))
  >> ERASE(OR(X,ISSAME(X),X2)) & SUCCEED(OR(X)) & NEGATE(1);
F43: "ONE CONT D" = ONE(SUCCEED(OR(X))) & MATCH(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  >> MATCH(OR(X,ISSAME(X),X2)) & ONE(SUCCEED(OR(X))) & NEGATE(1);
F44: "ONE CONT L" = ONE(SUCCEED(OR(X))) & LOCEXTN(X,OR(X))
  & NOT(EXISTS(X,OR(X))) & MATCHES(X,OR(X))
  >> LOCEXTN(X,OR(X)) & ONE(SUCCEED(OR(X))) & NEGATE(1);
F45: "ONE CONT R" = ONE(SUCCEED(OR(X))) & MATCHES(X,OR(X))
  & MATCHES(X,OR(X)) & ONE(SUCCEED(OR(X))) & NEGATE(1);
F46: "ONE CONT S" = ONE(SUCCEED(OR(X))) & TESTONE(X,OR(X))
  >> TESTONE(X,OR(X)) & ONE(SUCCEED(OR(X))) & NEGATE(1);
F47: "LINES" = ONE(SUCCEED(OR(X))) & ONE(SUCCEED(OR(X))) & NEGATE(1);
F48: "ERS OBJ" = ERASE(OR(X,ISSAME(X),X2)) & HASTOPONE(X)
  >> ERASE(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F49: "ERS OBJ N" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> ERASE(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F50: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & HASTOPONE(X)
  >> NEGATE(ALL);
F51: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F52: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F53: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F54: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F55: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F56: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F57: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F58: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F59: "ERS OBJ NV" = ERASE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(ALL);
F60: "FILE DES ASC" = FILE(OR(X,ISSAME(X),X2)) & DEPRHS(OR(X))
  & ASSIGN(OR(X,ISSAME(X),X2)) & ASSIGN(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & ASSIGN(OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & GREAT(M)
  >> ASSIGN(OR(X,ISSAME(X),X2)) & EXTONE(OR(X,ISSAME(X),X2)) & NEGATE(1);
  & TARGET DESIRED ASSIGNMENT
  >> "DA-3: "SAMPLE DA MET" = ASSIGN(OR(X,ISSAME(X),X2)) & SATISF(OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & SATISF(OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & SATISF(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & ASSIGN(OR(X,ISSAME(X),X2))
  & NOT(VE(X,OR(X))) & VE(X,OR(X))
  & NOT(VE(X,OR(X))) & VE(X,OR(X))
  >> ISSAME(OR(X,ISSAME(X),X2)) & NEGATE(ALL);
F61: "OLD DA" = ISSAME(OR(X,ISSAME(X),X2)) & EXTONE(OR(X,ISSAME(X),X2))
  & DEPRHS(OR(X,ISSAME(X),X2))
  >> MORE(OR(X,ISSAME(X),X2)) & HASTOPONE(X) & NEGATE(ALL);
F62: "MORE DA" = MORE(OR(X,ISSAME(X),X2)) & EXTONE(OR(X,ISSAME(X),X2))
  >> FILE(OR(X,ISSAME(X),X2)) & NEGATE(1);
F63: "MORE DA" = MORE(OR(X,ISSAME(X),X2)) & NOT(EXISTS(X,OR(X)))
  >> NEGATE(1);
F64: "EXT ON DA MET" = EXTONE(OR(X,ISSAME(X),X2)) & NEGATE(1);
F65: "DA MET COL" = EXTONE(OR(X,ISSAME(X),X2)) & ASSIGN(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & ASSIGN(OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & SATISF(OR(X,ISSAME(X),X2))
  & NOT(EXISTS(X,OR(X))) & ASSIGN(OR(X,ISSAME(X),X2))
  & SATISF(OR(X,ISSAME(X),X2)) & SATISF(OR(X,ISSAME(X),X2))
  >> COLONE(OR(X,ISSAME(X),X2))

```

14-00000

```

IF CONTINUE DESIRED, ADD PROG FOR EXPLICIT FAIL &
M37N "GET COMP." = GENDESASG(GOPDACL.D) & HASLPCOMPOND.P)
& NOT(EXISTSP2) & HASLPCOMPOND.P) & MAKE UNIQUE &
& SATISF IIS7P2P2 LEHL P1)
-> GENDESASG(GOPDACL.D) & NEGATE(1);
M38G: "GET COMP." = GENDESASG(GOPDACL.D)
& NOT(EXISTSP) & HASLPCOMPOND.P)
-> GETLPCOMPOND.P) & GENDESASG(GOPDACL.D) & NEGATE(1);
M38N "GEN DES ASG" = GENDESASG(GOPDACL.D) & HASVARE.VAR)
& HASLPCOMPOND.P) & VARDOHA(MVAR.P) & FIES MULTPLY &
-> CHECK(MVDA) & FILEDESASG(DA) & FEASASG(OPDA)
& ASSIGN(MDA VAR.P)
M38A: "GEN DES ASG AND" = GENDESASG(GOPDACL.D) & HASVARE.VAR)
& CHANGT SYM(VAR) & HASMOVECOMPONDOP.C.V1.V2)
& VARDOHA(MVAR.P)
-> ERASE(PC1) & FILEDESASG(DA) & FEASASG(OPDA)
& ASSIGN(MDA VAR.V2) & M(GATE(1));
M37: "TOS LC" = (BASE(PC1) & HASLPCOMPOND.P)
& NOT(EXISTSG(X1.X2.X3.X4.X5))
& (GENDESASG(X1.X2.X3.X4.X5))
& NOT(EXISTSG(X1.X2.X3.X4.X5) & SELECTDESASG(X1.X2.X3.X4.X5))
-> NEGATE(ALL);
M37B: "TOS LC RE AS" = (BASE(PC1) & GENDESASG(X1.X2.X3.X4.X5))
-> GENDESASG(X1.X2.X3.X4.X5) & NEGATE(1);
M37S: "TOS LC RE AS" = (BASE(PC1) & SELECTDESASG(X1.X2.X3.X4.X5))
-> SELECTDESASG(X1.X2.X3.X4.X5) & NEGATE(1);
M38: "GEN DES ASG" = CHECK(MVDA) & GENDESASG(GOPDACL.D)
& HASVARE.VAR) & HASVARE(MVAR.P) & HASLPCOMPOND.P)
& VARDOHA(MVAR.P)
& SATISF IIS7P2P2 LEHL P1) & NOT( & GREAT 0)
-> ERASE(PC1) & FILEDESASG(DA) & FEASASG(OPDA)
& ASSIGN(MDA VAR) & NEGATE(1,2);
& ASSUME'S NAME'S ALWAYS IMPLES CHANGE. THIS WANT NON-0
COULD BE GENERALIZED EASILY IF NECESS. &
& MAY FIRE MULTIPLY SPECIFYING A DESIRABLE SET OF VALUES &
M38P: "GEN ASG" = CHECK(MVDA) & GENDESASG(GOPDACL.D)
& HASVARE.VAR) & HASVARE(MVAR.P) & NOT(HASLPCOMPOND.P)
& HASDESASG(DA) & ASSIGN(MDA X1.X2)
& FEASASG(OPDA) & FILEDESASG(DA)
-> ERASE(PC1) & NEGATE(1,2,3,4,5);
& ASSUME'S AT LEAST ONE NON-NUM COMPON OF LOC PROG TO GET ASSIGNM &
M38P: "GEN DES ASG" = CHECK(MVDA) & GENDESASG(GOPDACL.D)
& NOT(EXISTSVAR.P) & HASVARE.VAR) & HASVARE(MVAR.P)
-> ERASE(PC1) & NEGATE(1,2);

```

& FOR ALL TRANS'S BE SURE ALL FEASASG'S OMITTED
BEFORE TRYAPP EXAMINED &

& TRYAPP: APPLY, OR EXTRACT OP DIFFR &

```

M38Q: "TRY APPLY" = TRYAPP(GOP) & HASMEWFEAS(GOPADA)
& NOT(EXISTSG(X1.X2.X3)) & TRYAPP(GOP) & VIE(GOPD2)
& HASMEWFEAS(GOPD2) & VIE(GOPD2)
& NOT(EXISTSG(A2.A3.A7) & HASMEWFEAS(GOPD2.A2.A3.A7)
& SATISF IIS7P2P2 LEHL P1) & GREAT M)
& NOT(EXISTSG(A2.A3) & HASMEWFEAS(GOPD2.A2.A3)
& SATISF IIS7P2P2 LEHL A2))
& HASACTUAL OR(GOP)
& NOT(EXISTSG(A2) & FEASASG(OPD2))
-> APPLY(OPADA) & APPLY(CR(GOPDAA) & NEGATE(1,2);
M38Q: "TRY APPLY ME" = TRYAPP(GOP) & TRYAPP(GOP)
& SATISF IIS7P2P2 LEHL P1)
& HASMEWFEAS(GOPD2) & HASMEWFEAS(GOPD2.X1.X2.X3)
& NOT(EXISTSG(X1.X2.X3) & TRYAPP(GOP)
& SATISF IIS7P2P2 LEHL P1)
& HASMEWFEAS(GOPD2.X1.X2.X3)
-> TRYAPP(GOP) & TRYAPP(GOP) & NEGATE(1,2);
M38Q: "RE AS FEASASG" = HASMEWFEAS(GOPD2.ADA) & FEASASG(OPD2)
& TRYAPP(GOP)
-> FEASASG(OPD2) & TRYAPP(GOP) & NEGATE(1);
M38Q: "UN HOLD TRYAPP" = TRYAPP(GOP) & TRYAPP(GOP) & NEGATE(1);
& IF FEASASG NOT CONSUMED, WILL HAVE LOOP 1;
M38: "APPLY SUC" = APPLY(CR(GOPDAA) & APPLY(SLE(TOPD1))
-> FILEOR(GOPD1) & SUCCESS(GOPD1) & NEGATE(ALL) & NOT TRYAPP(GOP)
M38: "EVAL OP DIFFR" = APPLY(CR(GOPDAA) & APPLY(DIFFR(V1.V2)OP)
-> DIFFRVAL(V1.V2)
& TRYAPP(DIFFR(TUP(GOPDAA.V1.V2) & TRYAPP(GOP)
& NEGATE(ALL);
M38: "DIFFR DIFFR" = TRYAPP(DIFFR(TUP(GOPDAA.V1.V2)
& DIFFRVAL(RESULT(V1.V2)
& HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& SATISF IIS7P2P2 LEHL P1)
& NOT(EXISTSG(X1.X2.X3)

```

```

& TRYAPP(DIFFR(TUP(GOPDAA.V1.V2) & NEGATE(ALL);
& DIFFRVAL(RESULT(V1.V2)
& SATISF IIS7P2P2 LEHL P1)
-> TRYAPP(GOP) & HASOP(DIFFRASC(GOPDAA.V1.V2) & NEGATE(ALL);
M38: "DIFFR DIFFR" = TRYAPP(DIFFR(TUP(GOPDAA.V1.V2)
& DIFFRVAL(RESULT(V1.V2)
& DIFFRVAL(RESULT(V1.V2)
& NOT(EXISTSG(X1.X2.X3)
& HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& NOT(EXISTSG(X1.X2.X3)
& TRYAPP(DIFFR(TUP(GOPDAA.V1.V2) & NEGATE(ALL);
M38: "DIFFR DIFFR" = TRYAPP(DIFFR(TUP(GOPDAA.V1.V2)
& DIFFRVAL(RESULT(V1.V2)
& DIFFRVAL(RESULT(V1.V2)
& HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& NOT SATISF IIS7P2P2 LEHL P1)
-> TRYAPP(GOP) & HASOP(DIFFRASC(GOPDAA.V1.V2) & NEGATE(1,2);
M38: "APPLY FAIL" = APPLY(CR(GOPDAA) & NOT(EXISTSG(X1.X2.X3)
& NOT(EXISTSG(V1.V2) & APPLY(DIFFR(V1.V2)OP)
-> ERASE(AMP(GA) & TRYAPP(GOP) & TRYAPP(GOP) & NEGATE(1);
M38: "APPLY BASE" = (BASE(AMP(GA) & APPLY(OPADA) & ASSIGN(MA.V1.V2)
-> NEGATE(ALL);
M38: "SEL DIFFR" = TRYAPP(GOP) & HASOP(DIFFRASC(GOPDAA.V1.V2)
& NOT(EXISTSG(X1.X2.X3) & HASMEWFEAS(GOPDAA.V1.V2)
& NOT(EXISTSG(X1.X2) & APPLY(LOC PROG(X1.X2)
& NOT(EXISTSG(A2) & HASOP(DIFFRASC(GOPDAA.V1.V2)
& VIE(GA.P1))
& NOT(EXISTSG(D2.A2.V2.W1.W2)
& HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& SATISF IIS7P2P2 LEHL P1)
& NOT(EXISTSG(D2.A2.V2.W1.W2)
& HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& SATISF IIS7P2P2 LEHL A2))
& PICKS FASTEST HARDEST OP DIFFR USING LENGTHENED FOR TIES &
-> TRYAPP(RESULT(GOPDAA.V1.V2) & NEGATE(1,2)
& NOT TRYAPP(GOP);
M38: "SEL IDENT" = TRYAPP(GOP) & HASOP(DIFFRASC(GOPDAA.V1.V2)
& NOT(EXISTSG(X1.X2.X3) & HASMEWFEAS(GOPDAA.V1.V2)
& NOT(EXISTSG(X1.X2) & APPLY(LOC PROG(X1.X2)
& HASOP(DIFFRASC(GOPDAA.V1.V2)
& SATISF IIS7P2P2 LEHL P1)
& NOT(EXISTSG(A2) & HASOP(DIFFRASC(GOPDAA.V1.V2)
& SATISF IIS7P2P2 LEHL A2))
-> HASOP(DIFFRASC(GOPDAA.V1.V2) & NEGATE(1,2);
M38: "NO DIFFR" = TRYAPP(GOP) & NOT(EXISTSGOPDAA.V1.V2)
& HASOP(DIFFRASC(GOPDAA.V1.V2)
& NOT(EXISTSGOP) & TRYAPP(GOP)
-> FAIL(GOP) & METHODSETH(GOP) & NEGATE(1);
M38: "NO DIFFR TA" = TRYAPP(GOP) & NOT(EXISTSGOPDAA.V1.V2)
& HASOP(DIFFRASC(GOPDAA.V1.V2)
& TRYAPP(GOP)
& NOT(EXISTSGADA.P) & HASMEWFEAS(GOPDAA)
-> FAIL(GOP) & METHODSETH(GOP) & NEGATE(ALL);
M38: "TRYAPP RES" = TRYAPP(RESULT(GOPDAA.V1.V2) & REDUCE(ORAL(GOP)
& HASACTUAL OR(GOP) & HASDIFF(LOC V3)
-> EXISTSG(GOP) & FILE GOAL(GOP) & EVAL GOAL(GOP) & ISAPPLY GOAL(GOP)
& HASDESASG(DA) & HASOP(DIFFRASC(GOPDAA.V1.V2) & HASOP(GOP)
& HASACTUAL OR(GOP) & HASDIFF(LOC V3) & HASSUPERGOAL(GOP)
& NEGATE(1);
M38: "RE TRY ASG" = REDUCE(MTH(GOP) & HASMEWFEAS(GOPDAA) & RETRYED
-> TRYAPP(GOP) & NEGATE(1,2);
M38: "RE TRY OPD" = REDUCE(MTH(GOP) & RETRYED
& NOT(EXISTSGADA.P) & HASMEWFEAS(GOPDAA)
& HASDESASG(DA)
-> TRYAPP(GOP) & NEGATE(1,2);
M38: "RE TRY NOTHING" = REDUCE(MTH(GOP) & RETRYED
& NOT(EXISTSGADA) & HASDESASG(DA)
-> REDUCE(MTH(GOP) & NEGATE(1,2);
& MOVE OPERATOR METHOD MAY ALREADY HAVE OP DIFFR &
M38: "MOVE OP" = MOVE(MTH(GOP) & NOT RETRYED) & HASOP(DIFFRASC(GOPDAA.V1.V2)
& HASDIFF(LOC V3)
-> SPOLIT(MTH(GOP) & V1.V2.V3) & NEGATE(1);
& V3 FROM M38 IS A DUMMIE NOT SO FOR M37 THOUGH &
M38: "RE W RED APP" = SPOLIT(MTH(GOP) & V1.V2.V3) & HASDESASG(DA)
& HASDIFF(LOC V3) & HASACTUAL OR(GOP)
-> EXISTSG(GOP) & FILE GOAL(GOP) & EVAL GOAL(GOP) & HASUPERGOAL(GOP)

```

```

      & HASDIFF(CO.V) & IREDUCE(CO.V) & HASDIFF(CO.V)
      & HASACTUAL(OR.O) & NEXT(OR.APPLY(CO.DA.V) & NEGATE(1))

% MOVE OP: MAKE FEASIBLE AND TRY APPLY %
% M57 - M59 ARE LIKE M47 - M49 %

M52: "MOVE OP - DIFF" = MOVE(OP:ME THING) & NOT TRY(C) & HASOP(CO)
      & NOT(EXISTS(V1.V2) & HASOPDIFF(CO.V1.V2))
      & HASDIFF(CO.DA)
      & FEASIBLE(CO.DA) & TRYAPPLY(CO) & NEGATE(1)
M57: "TRYAPP RES" = TRYAPPRESLA(TG(CO.DA.V1.V2) & ISAPPLY(CO)
      & HASDIFF(CO.V3)
      & SPOLITH(HAPPY(CO.V1.V2.V3) & NEGATE(1.3) & HASDIFF(CO.V)
      & TRYAPPLY(CO) & NEGATE(1.3)
M58: "RETRY ASG" = MOVE(OP:ME THING) & HASWFEASIBLE(CO.DA) & RETRY(C)
      & TRYAPPLY(CO) & NEGATE(1.3)
M59: "RETRY OPD" = MOVE(OP:ME THING) & RETRY(C)
      & NOT(EXISTS(CO.DA) & HASWFEASIBLE(CO.DA)
      & TRYAPPLY(CO) & NEGATE(1.3)

% MATCHDIFF METHOD %

R0: "MATCHDIFF 1ST" = MATCHDIFF(CO.D1.D2) & HASOP(CO)
      & HASOP(CO)
      & MATCHDIFF(CO.D1.D2) & NEGATE(1)
R1: "M2" = MATCHDIFF(CO.D1.D2) & LINKS(L1.M1) & LINKS(L2.M2)
      & NOT(EXISTS(CO.D1.D2) & MATCHRES(L1))
      & MATCHDIFF(CO.D1.D2)
      & CAN'T NEGATE ALL MATCHDIFF'S BECAUSE DIFFERENT PRODS FIRE AT
      SAME NODE: MUST LEAVE SOME AROUND
      TO MAKE SURE (BASE LATER WILL FIND SOME)
R3: "M2 BAD VAL" = MATCHDIFF(CO.D1.D2) & HASVAL(M1.V1) & HASVAL(M2.V2)
      & VNEQ(V1.V2)
      & EXISTS(CO) & LOCEXTN(CO.D1.V1.V2.D1.D2) & NEGATE(1)
R6: "M2 UNDEF M1" = MATCHDIFF(CO.D1.D2) & LINKS(L1.M1)
      & NOT(EXISTS(M1) & LINKS(L1.M1))
      & MATCHDIFF(CO.D1.D2)
R8: "M2 UNDEF M2" = MATCHDIFF(CO.D1.D2) & LINKS(L1.M1)
      & NOT(EXISTS(M2) & LINKS(L1.M2))
      & MATCHDIFF(CO.D1.D2)
R9: "M2 UNDEF V1" = MATCHDIFF(CO.D1.D2) & HASVAL(M1.V1)
      & NOT(EXISTS(V1) & HASVAL(M1.V1))
      & EXISTS(CO) & LOCEXTN(CO.D1.V1.D1.D2) & NEGATE(1)
R7: "M2 UNDEF V2" = MATCHDIFF(CO.D1.D2) & HASVAL(M1.V1)
      & NOT(EXISTS(V2) & HASVAL(M1.V2))
      & EXISTS(CO) & LOCEXTN(CO.D1.V1.D1.D2) & NEGATE(1)

% EXTRACT LOCATION OF DIFF AND LIMIT RESULTS %

R0: "LOC EXTN" = LOCEXTN(CO.D1.V1.V2.D1.D2) & LINKS(L1.M1)
      & NOT(EXISTS(CO) & HASOP(CO))
      & LOCEXTN(CO.D1.V1.V2.D1.D2) & HASLINK(D1.M) & NEGATE(1)
R0: "LOC EXTN TOP" = LOCEXTN(CO.D1.V1.V2.D1.D2) & LINKS(L1.M1)
      & HASOP(CO) & NOT(EXISTS(CO) & MATCHDIFF(CO))
      & FILE(LOC(CO.D1) & HASLINK(D1.M) & RESULTSET(TO(D1.V1.V2) & NEGATE(1)
      & MATCHDIFF(NON-MATCHDIFF) COULD NOW EASILY RETURN
      RESULTS FROM 2 MATCHES AT ONCE
      NEED ONLY AND 2 OBJECT ARGS TO RESULTSETIP %
R10: "M2 RESULT P" = RESULTSET(TO(D1.V1.V2) & HASNAME(D1))
      & MATCHRES(L1.V1.V2) & NEGATE(ALL)
R11: "LOC EXTN TOP 1" = LOCEXTN(CO.D1.V1.V2.D1.D2) & LINKS(L1.M1)
      & HASOP(CO) & MATCHDIFF(CO)
      & MATCHRES(L1.V1.V2.D1.D2) & HASLINK(D1.M) & NEGATE(1)

END:

```

EXPR OPER: BEGIN % LITLITY BELTINES PAGE 8 %

% PRODUCTIONS FOR TRANSFORMATIONS %

% ADDLINK(DA.V1.V2) ARE NUMERIC (POSITIVE), INCREASING WITH
DEPTH AWAY FROM TOPNODE, UNTIL "VALUE" REACHED, WHICH MUST
BE NON-NUMERIC VIA GROUPS ADDLINK'S INTO SETS, FOR
DOING A PARTICULAR NODE PATH: L IS LINK: ADDLAST REEPS
POSITION OF LAST ADDITION IN PATH %
% COULD HAVE "VALUE" BE POSITIVE NUMERIC BY ADOPTING NEGATIVE-
INTEGER CONVENTION %

```

T1: "ADDLINK 1" = ADDLINK(DA.V1) & SATISF(RESV.VQ 1) & HASOP(CO)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & EXISTS(CO) & ADDLAST(M1) & LINKS(L1.M1) & NEGATE(1)
T2: "ADDLINK 1" = ADDLINK(DA.V1) & SATISF(RESV.VQ 1) & HASOP(CO)
      & LINKS(L1.M1)
      & ADDLAST(M1) & NEGATE(1)
T3: "ADDLINK 1" = ADDLINK(DA.V1) & ADDLAST(M1) & SATISF(RESV.VQ 1)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & SATISF(RESV.V2.V2 %LESS V1)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & EXISTS(CO) & ADDLAST(M1) & LINKS(L1.M1) & NEGATE(1.2)
T4: "ADDLINK 1" = ADDLINK(DA.V1) & ADDLAST(M1) & SATISF(RESV.VQ 1)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & SATISF(RESV.V2.V2 %LESS V1)
      & LINKS(L1.M1)
      & ADDLAST(M1) & NEGATE(1.2)
T5: "ADDLINK 1" = ADDLINK(DA.V1) & NOT SATISF(RESV.VQ 1)
      & ADDLAST(M1)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & SATISF(RESV.V2.V2)
      & EXISTS(CO) & LINKS(L1.M1) & HASVAL(M1.V1) & NEGATE(1.3)
T6: "ADDLINK 1" = ADDLINK(DA.V1) & NOT SATISF(RESV.VQ 1)
      & NOT(EXISTS(V1) & ADDLAST(M1)) & HASOP(CO)
      & NOT(EXISTS(V2) & LINKS(L1.M1))
      & SATISF(RESV.V2.V2)
      & EXISTS(CO) & LINKS(L1.M1) & HASVAL(M1.V1) & NEGATE(1)

% SET OF REMLINKS, WITH 1ST & 2ND ARG IN COMMON
% SPECIFIC REMOVAL, ALONG PATH OF SET
% SINCE REMLINK -- REMOVE IT AND EVERYTHING UNDER IT
% REMLAST RECORDS POSITION OF LAST REMOVAL %

T10: "REMLINK ALL TOP" = REMLINK(CO.D1) & HASOP(CO) & LINKS(L1.M1)
      & NOT(EXISTS(V2) & REMLINK(CO.D1) & VNEQ(L1.L2))
      & LINKS(L1.M1)
      & EXISTS(CO) & REMLINK(CO.D1) & REMLAST(M1) & NEGATE(1.3)
T11: "REMLINK SPEC TOP" = REMLINK(CO.D1) & HASOP(CO) & LINKS(L1.M1)
      & REMLINK(CO.D1) & VNEQ(L1.L2)
      & REMLAST(M1) & NEGATE(1)
T12: "REMLINK ALL AND" = REMLINK(CO.D1) & REMLAST(M1) & LINKS(L1.M1)
      & NOT(EXISTS(V2) & REMLINK(CO.D1) & VNEQ(L1.L2))
      & LINKS(L1.M1)
      & EXISTS(CO) & REMLINK(CO.D1) & REMLAST(M1) & NEGATE(1.2.3)
T13: "REMLINK VAL" = REMLINK(CO.D1) & REMLAST(M1) & LINKS(L1.M1)
      & HASVAL(M1.V1)
      & NEGATE(ALL)
T14: "REMLINK AND C" = REMLINK(CO.D1) & REMLAST(M1) & LINKS(L1.M1)
      & REMLINK(CO.D1) & VNEQ(L1.L2)
      & REMLAST(M1) & NEGATE(1.2)

% ARGS OF INCR LINK & DECR LINK AS FOR ADDLINK
% EXCEPT NON-POSITIVE V NOW STANDS FOR AMOUNT TO INCR/DECR %

T20: "INCR LINK 1" = INCR LINK(D1.V1) & SATISF(RESV.VQ 1)
      & HASOP(CO) & LINKS(L1.M1)
      & INCR AS(T1.M1) & NEGATE(1)
T21: "INCR LINK 1" = INCR LINK(D1.V1) & INCR AS(T1.M1)
      & SATISF(RESV.V %GREAT 0)
      & NOT(EXISTS(V2) & INCR LINK(D1.V2)
      & SATISF(RESV.V2 %GREAT 0)
      & SATISF(RESV.V2.V2 %LESS V1)
      & LINKS(L1.M1)
      & INCR AS(T1.M1) & NEGATE(1.2)
T22: "INCR LINK 1" = INCR LINK(D1.V1) & NOT SATISF(RESV.V %GREAT 0)
      & INCR AS(T1.M1) & LINKS(L1.M1) & HASVAL(M1.V1)
      & NEGATE(ALL) & HASVAL(M1.V1)
      & INCR LINK(V1) = INCR LINK(D1.V1) & NOT SATISF(RESV.V %GREAT 0)
      & HASOP(CO) & LINKS(L1.M1) & HASVAL(M1.V1)
      & NEGATE(1.3) & HASVAL(M1.V1)

T30: "DECR LINK 1" = DECR LINK(D1.V1) & SATISF(RESV.VQ 1)
      & HASOP(CO) & LINKS(L1.M1)
      & DECR AS(T1.M1) & NEGATE(1)
T31: "DECR LINK 1" = DECR LINK(D1.V1) & DECR AS(T1.M1)
      & SATISF(RESV.V %GREAT 0)
      & NOT(EXISTS(V2) & DECR LINK(D1.V2)
      & SATISF(RESV.V2 %GREAT 0)
      & SATISF(RESV.V2.V2 %LESS V1)
      & LINKS(L1.M1)
      & DECR AS(T1.M1) & NEGATE(1.2)
T32: "DECR LINK 1" = DECR LINK(D1.V1) & NOT SATISF(RESV.V %GREAT 0)
      & DECR AS(T1.M1) & LINKS(L1.M1) & HASVAL(M1.V1)
      & NEGATE(ALL) & HASVAL(M1.V1)

```

14-00

OPEN

Appendix B. PRODUCTIONS FOR TOWER

```

BEGIN      % OPEN MONKEY AND BANANAS FORMATION %
ENDP OPBND: BEGIN      % ACQUIRE (OPBND); PMACTOP (OPBND);

Q1: "M INIT" = M INIT (M)
  -> EXIST (M) & OPBND (M) & FILE GOAL (M) & FILE OBJECT (INITIAL OBJECT)
    & EVAL GOAL (G) & IS TRANSFORM GOAL (G) & HAS SUPER GOAL (G, TOP)
    & OBJECT (INITIAL OBJECT)
    & MONKEY PLACE PLACE 1 BOX PLACE PLACE 2
    & HAS ACTION OR GOAL (INITIAL OBJECT)
    & HAS DESCRIPTION (DESCRIPTION OBJECT)
    & IS DESCRIBED (DESCRIPTION OBJECT)

Q2:
  & OBJECT (TABLE COMM, MONKEY PLACE OPSET 1)
    & MONKEY HAND GET BANANAS
    & BOX PLACE MOVE BOX
  & IS SET (OPSET 1) & IS SET (CL INB, OPSET 1)
  & IS SET (MOVE BOX, OPSET 1) & IS SET (WALK, OPSET 1)

Q3:
  % THE FOLLOWING IS FIRST & NEWELL'S TABLE OF COMB'S %
  & OBJECT (TABLE COMM, MONKEY PLACE OPSET 2)
    & MONKEY HAND OPSET 2
    & BOX PLACE OPSET 2
  & IS SET (OPSET 2) & IS SET (CL INB, OPSET 2)
  & IS SET (CL BANANAS, OPSET 2) & IS SET (MOVE BOX, OPSET 2)
  & IS SET (WALK, OPSET 2)

  & OBJECT (DIFFER,
    & MONKEY PLACE 1 MONKEY HAND 3 BOX PLACE 2)
  & IS MOVE OP (CL INB) & IS MOVE OP (CL BANANAS)
  & IS MOVE OP (MOVE BOX) & IS MOVE OP (WALK)
  & VAR DOMAINS (CL INB, CL BANANAS) & VAR DOMAINS (CL INB, CL BANANAS)
  & VAR DOMAINS (MOVE TO, PLACE 1) & VAR DOMAINS (MOVE TO, PLACE 2)
  & VAR DOMAINS (MOVE TO, UNDER BANANAS)
  & VAR DOMAINS (WALK TO, PLACE 1) & VAR DOMAINS (WALK TO, PLACE 2)
  & VAR DOMAINS (WALK TO, UNDER BANANAS)
  & CHANGES VAL (WALK TO) & CHANGES VAL (MOVE TO)
  & CHANGES VAL (CL INB)
  & HAS MOVE COMPON (CL INB, CL, APP, APP) & HAS VAR (CL, CL INB)
  & HAS MOVE COMPON (CL BANANAS, CL, INB, CL, BANANAS)
  & HAS VAR (CL, CL BANANAS)
  & HAS MOVE COMPON (MOVE BOX, CL, APP, APP)
  & HAS VAR (CL, CL MOVE BOX)
  & HAS MOVE COMPON (WALK, CL, APP, APP) & HAS VAR (CL, CL WALK)

Q4: "APPLY CL INB" = APPLY OP (OP A OR) & SATISF (IS OP (EQ CL INB)
  & ASSIGN (A, X1, X2) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & LINK (L2 IN IN) & SATISF (IS L2 EQ BOX PLACE)
  & HAS VAL (N2, V1) & TRAC (IN IN)
  -> EXIST (M) & TRACING (TRAC PRINT (M, "APPLY OP, TO OR, GET D, X))
  & COPY OR (OR IN) & COPY IN (IN, CL INB, OR IN)
  & APPLY RESULT (OP D) & NEGATE (1, 3);

Q5: "APPLY GET BANANAS" = APPLY OP (OP A OR)
  & SATISF (IS OP (EQ GET BANANAS) & ASSIGN (A, X1, X2)
  & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ BOX PLACE) & HAS VAL (N2, V1)
  & SATISF (IS L2 EQ UNDER BANANAS) & LINK (L2 IN IN)
  & SATISF (IS L2 EQ MONKEY PLACE) & HAS VAL (N2, V2)
  & SATISF (IS L2 EQ CL INB) & TRAC (IN IN)
  -> EXIST (M) & TRACING (TRAC PRINT (M, "APPLY OP, TO OR, GET D, X))
  & COPY OR (OR IN) & COPY IN (IN, CL INB, MONKEY HAND BANANAS)
  & APPLY RESULT (OP D) & NEGATE (1, 3);

Q6: "APPLY MOVE BOX" = APPLY OP (OP A OR) & SATISF (IS OP (EQ MOVE BOX)
  & ASSIGN (A, M TO LOC) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & LINK (L2 IN IN) & SATISF (IS L2 EQ BOX PLACE)
  & HAS VAL (N2, V1) & VAR (V1, LOC) & TRAC (IN IN)
  -> EXIST (M) & TRACING (TRAC PRINT (M,
    & "APPLY OP, TO OR, GET D, M TO LOC, X))
  & COPY OR (OR IN) & COPY IN (IN, CL INB, LOC)
  & COPY IN (IN, CL2 LOC) & APPLY RESULT (OP D) & NEGATE (1, 3);

Q7: "APPLY WALK" = APPLY OP (OP A OR) & SATISF (IS OP (EQ WALK)
  & ASSIGN (A, M TO LOC) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & TRAC (IN IN)
  & HAS VAL (N2, V1) & VAR (V1, LOC) & DON'T WALK TO SAME PLACE %
  -> EXIST (M) & TRACING (TRAC PRINT (M,
    & "APPLY OP, TO OR, GET D, M TO LOC, X))
  & COPY OR (OR IN) & COPY IN (IN, CL INB, LOC) & APPLY RESULT (OP D)
  & NEGATE (1, 3);

Q8: "DIFF CL INB" = APPLY OP (OP A OR) & SATISF (IS OP (EQ CL INB)

```

```

  & ASSIGN (A, X1, X2) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & LINK (L2 IN IN) & SATISF (IS L2 EQ BOX PLACE)
  & HAS VAL (N2, V2) & VAR (V1, V2) & LOCATIONS DON'T MATCH %
  -> EXIST (M) & FILE (LOC PROC) & APPLY OP (OP A OR) & NEGATE (1, 3);
  & HAS IN (IN, LOC) & NEGATE (1, 3);

Q9: "DIFF GET BANANAS MON" = APPLY OP (OP A OR)
  & SATISF (IS OP (EQ GET BANANAS) & ASSIGN (A, X1, X2)
  & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ BOX PLACE) & HAS VAL (N2, V1)
  & SATISF (IS L2 EQ UNDER BANANAS) & LINK (L2 IN IN)
  & SATISF (IS L2 EQ MONKEY PLACE) & HAS VAL (N2, V2)
  & NOT SATISF (IS L2 EQ CL INB) & DON'T WALK TO SAME PLACE %
  -> EXIST (M) & FILE (LOC PROC) & APPLY OP (OP A OR) & NEGATE (1, 3);
  & HAS IN (IN, LOC) & NEGATE (1, 3);

Q10: "DIFF GET BANANAS MON" = APPLY OP (OP A OR)
  & SATISF (IS OP (EQ GET BANANAS) & ASSIGN (A, X1, X2)
  & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ BOX PLACE) & HAS VAL (N2, V1)
  & SATISF (IS L2 EQ UNDER BANANAS) & LINK (L2 IN IN)
  & SATISF (IS L2 EQ MONKEY PLACE) & HAS VAL (N2, V2)
  & NOT SATISF (IS L2 EQ CL INB) & DON'T WALK TO SAME PLACE %
  -> EXIST (M) & FILE (LOC PROC) & APPLY OP (OP A OR) & NEGATE (1, 3);
  & HAS IN (IN, LOC) & NEGATE (1, 3);
  % NO DIFF POSSIBLE FOR WALK %

Q11: "OP DIFF NAMED" = APPLY OP (OP A OR) & SATISF (IS OP (EQ MOVE BOX)
  & ASSIGN (A, X1, X2) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & LINK (L2 IN IN) & SATISF (IS L2 EQ BOX PLACE)
  & HAS VAL (N2, V2) & VAR (V1, V2) & LOCATIONS DON'T MATCH %
  & VAR (V2, V1) & DON'T WALK TO SAME PLACE %
  -> EXIST (M) & FILE (LOC PROC) & APPLY OP (OP A OR) & NEGATE (1, 3);
  & HAS IN (IN, LOC) & NEGATE (1, 3);
  % NO DIFF POSSIBLE FOR WALK %

Q12: "OP DIFF NAMED" = APPLY OP (OP A OR) & SATISF (IS OP (EQ MOVE BOX)
  & ASSIGN (A, X1, X2) & HAS TOP NODE (OR IN) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & LINK (L2 IN IN) & SATISF (IS L2 EQ BOX PLACE)
  & HAS VAL (N2, V2) & VAR (V1, V2) & LOCATIONS DON'T MATCH %
  & VAR (V2, V1) & DON'T WALK TO SAME PLACE %
  -> EXIST (M) & FILE (LOC PROC) & APPLY OP (OP A OR) & NEGATE (1, 3);
  & HAS IN (IN, LOC) & NEGATE (1, 3);
  % NO DIFF POSSIBLE FOR WALK %

Q13: "MATCH DESCRIBED OBJ" = MATCH DIFF (0, 0, 0, 0) & IS DESCRIBED (OBJ)
  & HAS TOP NODE (OR IN)
  & NOT (EXIST (N2, V1) & LINK (L IN IN)
  & SATISF (IS L 1 EQ MONKEY PLACE) & HAS VAL (N2, V1)
  & SATISF (IS L2 EQ BANANAS)
  -> EXIST (M) & FILE (LOC PROC) & HAS IN (IN, MONKEY HAND)
  & RESULT (GET UNDER BANANAS)
  & LEAVE MATCH DIFF TO BE (BASED, M2) %

END END.

-----

BEGIN      % PRODUCTIONS FOR TOWER OF HANOI, FOR OPEN %
ENDP OPBND: BEGIN      % ACQUIRE (OPBND); PMACTOP (OPBND);

Q1: "M INIT" = M INIT (M)
  -> EXIST (M) & OPBND (M) & FILE GOAL (M) & FILE OBJECT (INITIAL OBJECT)
    & FILE OBJECT (DESCRIPTION OBJECT) & EVAL GOAL (G)
    & IS TRANSFORM GOAL (G) & HAS SUPER GOAL (G, TOP)

  & OBJECT (INITIAL OBJECT, PEG 1 (DISH 1 YES DISH 2 YES)
    PEG 2 (DISH 3 YES)
  & OBJECT (DESCRIPTION OBJECT, PEG 1 (DISH 1 YES DISH 2 YES)
    PEG 2 (DISH 1 YES DISH 2 YES)
  & OBJECT (INITIAL OBJECT, PEG 1 (DISH 1 YES DISH 2 YES)
    DISH 3 YES DISH 4 YES) PEG 2 (DISH 1 YES DISH 2 YES)
  & OBJECT (DESCRIPTION OBJECT, PEG 1 (DISH 1 YES DISH 2 YES)
    PEG 2 (DISH 1 YES DISH 2 YES DISH 3 YES DISH 4 YES)

Q2:
  & HAS ACTION OR GOAL (INITIAL OBJECT)
  & HAS DESCRIPTION (DESCRIPTION OBJECT)
  & OBJECT (TABLE COMM,
    PEG 1
    (DISH 1 MOVE DISH 1 DISH 2 MOVE DISH 1 DISH 3 MOVE DISH 1
    DISH 4 MOVE DISH 1)
    PEG 2
    (DISH 1 MOVE DISH 1 DISH 2 MOVE DISH 1 DISH 3 MOVE DISH 1
    DISH 4 MOVE DISH 1)

```

[illegible]

END: END

[illegible]

0 HASBMT(COMPONENT) CROSSRIVER, CDS, YES, UNDET)
0 HASVAL(CDS, YES)
0 HASBMT(COMPONENT) CROSSRIVER, CDS, UNDET, YES
0 HASVAL(CDS, YES)

Q4: "APPLY CROSSRIVER" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER
0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)
0 LINKS(L2,N1) 0 LINKS(L2,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
1 BOAT CAPACITY GUARANTEED BY FEASASG'S 1
0 LINKS(CN7,M4) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N4,MFC)
0 LINKS(M2,N5) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N5,NFM)
0 LINKS(L1,N6)
0 LINKS(CAN,M7) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N7,NTC)
0 LINKS(M6,M8) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N8,NTM)
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
1 AN INVISIBLE CONSTRAINT IN OPS 1
0 NOT SATISF IESMFM(MFC,NTM) 1 FROM SIDE TESTS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 NOT SATISF IESMFM(MFC,NTM) 0 MFC TO SIDE TESTS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 TRAC(I,NO)

0 EXISTSD) 0 TRAC(I,NO) 0 TRAC(I,NO)
APPLYOP(OPADR), (A1D, L2, L1, L1, MM, NC, X)
0 COPYOR(OR,OR) 0 MEM(I,NO,1,2) 0 MEM(I,NO,1,2)
0 ADD(I,NO,1,1) 0 ADD(I,NO,1,2) 0 YES
0 INCR(I,NO,1,1) 0 INCR(I,NO,1,2) 0 C
0 INCR(I,NO,2,1) 0 INCR(I,NO,2,2) 0 M
0 DECR(I,NO,1,2) 0 DECR(I,NO,1,2) 0 C
0 DECR(I,NO,2,2) 0 DECR(I,NO,2,2) 0 M
0 APPLYRESULT(OP) 0 NEGAT(1,2,3,5)

Q5: "CROSSRIVER DIFFER BOAT" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER
0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)
0 LINKS(L1,N1) 0 LINKS(L1,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
0 NOT EXISTSD) 0 X1,X2,X3 0 APPLYDIFFER(TUPD) 0 P,X1,V1
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)
1 NO NEED TO CHECK X'S VALUES, SINCE ONLY ONE SUCH GENABLE 1
0 EXISTSD) 0 APPLYOP(OPADR) 0 APPLYOP(OPADR)
0 FILE(LOC,PROG) 0 APPLYDIFFER(TUPD) 0 P,LOC, YES
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)

Q6: "CROSSRIVER DIFFER FROMSIDE" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER
0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)
0 LINKS(L2,N1) 0 LINKS(L2,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
1 BOAT CAPACITY GUARANTEED BY FEASASG'S 1
0 LINKS(CN7,M4) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N4,MFC)
0 LINKS(M2,N5) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N5,NFM)
0 LINKS(L1,N6)
0 LINKS(CAN,M7) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N7,NTC)
0 LINKS(M6,M8) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N8,NTM)
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
1 AN INVISIBLE CONSTRAINT IN OPS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 NOT EXISTSD) 0 X1,X2,X3,X4 0 APPLYDIFFER(TUPD) 0 P,X1,X2
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)
1 NO NEED TO CHECK X'S VALUES, SINCE DIFF AT THAT LOC UNIQUE 1
0 EXISTSD) 0 APPLYOP(OPADR) 0 APPLYOP(OPADR)
0 FILE(LOC,PROG) 0 APPLYDIFFER(TUPD) 0 P,LOC, YES
1 WANT TO DEC NUMBER OF CANS AT FROMSIDE 1
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)

Q7: "CROSSRIVER DIFFER FROMSIDE" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER

0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)

1 THE ROLE OF L1 AND L2 ARE REVERSED IN THE FOLLOWING,
COMPARED TO Q4 1

0 LINKS(L1,N1) 0 LINKS(L1,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
1 BOAT CAPACITY GUARANTEED BY FEASASG'S 1
0 LINKS(CN7,M4) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N4,MFC)
0 LINKS(M2,N5) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N5,NFM)
0 LINKS(L1,N6)
0 LINKS(CAN,M7) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N7,NTC)
0 LINKS(M6,M8) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N8,NTM)
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
1 AN INVISIBLE CONSTRAINT IN OPS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 NOT EXISTSD) 0 X1,X2,X3,X4 0 APPLYDIFFER(TUPD) 0 P,X1,X2
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)
1 NO NEED TO CHECK X'S VALUES, SINCE DIFF AT THAT LOC UNIQUE 1
0 EXISTSD) 0 APPLYOP(OPADR) 0 APPLYOP(OPADR)
0 FILE(LOC,PROG) 0 APPLYDIFFER(TUPD) 0 P,LOC, YES
1 WANT TO DEC NUMBER OF CANS AT FROMSIDE 1
1 BUT FROMSIDE 1 TO SIDE REVERSED BECAUSE BOAT WRONG 1
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)

Q8: "CROSSRIVER DIFFER TO SIDE" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER
0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)
0 LINKS(L2,N1) 0 LINKS(L2,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
1 BOAT CAPACITY GUARANTEED BY FEASASG'S 1
0 LINKS(CN7,M4) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N4,MFC)
0 LINKS(M2,N5) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N5,NFM)
0 LINKS(L1,N6)
0 LINKS(CAN,M7) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N7,NTC)
0 LINKS(M6,M8) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N8,NTM)
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
1 AN INVISIBLE CONSTRAINT IN OPS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 NOT EXISTSD) 0 X1,X2,X3,X4 0 APPLYDIFFER(TUPD) 0 P,X1,X2
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)
1 NO NEED TO CHECK X'S VALUES, SINCE DIFF AT THAT LOC UNIQUE 1
0 EXISTSD) 0 APPLYOP(OPADR) 0 APPLYOP(OPADR)
0 FILE(LOC,PROG) 0 APPLYDIFFER(TUPD) 0 P,LOC, YES
1 WANT TO DEC NUMBER OF CANS AT TO SIDE, BUT THAT'S AFTER APPL IC:
EQUIV IS TO DEC NUMBER OF CANS AT FROMSIDE 1
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)

Q9: "CROSSRIVER DIFFER TO SIDE" = APPLYOP(OPADR)

0 SATISF IESOP(OPADR) CROSSRIVER
0 ASSIGN(A,MM) 0 SATISF IESMM(EQ 'MMIS')
0 ASSIGN(A,NC) 0 SATISF IESNC(EQ 'NCAN')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 ASSIGN(A,LL) 0 SATISF IESLL(EQ 'LO')
0 HASOPNOTE(ORJN1)
1 THE ROLE OF L1 AND L2 ARE REVERSED IN THE FOLLOWING,
COMPARED TO Q4 1
0 LINKS(L1,N1) 0 LINKS(L1,N2) 0 SATISF IESB(EQ 'BOAT')
0 HASVAL(N2,V1) 0 SATISF IESV1(EQ 'YES') 1 BOAT OR 1
1 BOAT CAPACITY GUARANTEED BY FEASASG'S 1
0 LINKS(CN7,M4) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N4,MFC)
0 LINKS(M2,N5) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N5,NFM)
0 LINKS(L1,N6)
0 LINKS(CAN,M7) 0 SATISF IESFC(EQ 'CAN') 0 HASVAL(N7,NTC)
0 LINKS(M6,M8) 0 SATISF IESMFM(EQ 'MIS') 0 HASVAL(N8,NTM)
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
0 NOT SATISF IESMFM(MFC,NTM) 0 GREAT MFC
1 AN INVISIBLE CONSTRAINT IN OPS 1
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 SATISF IESMFM(MFC,NTM) 0 MFC LESS MFC - C
0 NOT EXISTSD) 0 X1,X2,X3,X4 0 APPLYDIFFER(TUPD) 0 P,X1,X2
0 HAS(I,NO,1,2) 0 HAS(I,NO,1,2)
1 NO NEED TO CHECK X'S VALUES, SINCE DIFF AT THAT LOC UNIQUE 1

RMSUSLS F30 F32 F32 F34 F39
 COPYLAST
 LMSUSLS T81 T82
 NESTEDL T83
 RMSUSLS T80 T81 T81 T87
 COPYLIM
 LMSUSLS T40 T41 T47 T83
 NESTEDL T41 T47 T83
 RMSUSLS T40 T41 T47 T83
 COPYORI
 LMSUSLS C1 C7 C7 C4
 RMSUSLS Q4 C1 C1 C7 C7 C3 C4
 DECREFAST
 LMSUSLS T31 T32
 RMSUSLS T30 T31 T31 T32
 DECREFIM
 LMSUSLS T30 T31 T32 T33
 NESTEDL T31
 RMSUSLS Q4 T30 T31 T32 T33
 DIFFREVAL
 LMSUSLS D1
 RMSUSLS M21 M47 D1
 DIFFREVALRES1
 LMSUSLS D2
 RMSUSLS D1 D2
 DIFFREVALRES2
 LMSUSLS T13 D4 D5
 RMSUSLS D2 D3 D4 D5
 DIFFREVALRESLT
 LMSUSLS M22 M43 M43 M43
 NESTEDL M43 M43
 RMSUSLS M22 M43 M43 M43 D3 D4 D5
 ERASEAPP
 LMSUSLS M48
 RMSUSLS M48 M48
 ERASECHOICES
 LMSUSLS T37
 RMSUSLS T31 T32
 ERASECHOICESO
 LMSUSLS T37
 RMSUSLS T36 T37
 ERASECS
 LMSUSLS T87
 RMSUSLS T85 T87 T87 T82
 ERASECSP
 LMSUSLS T87
 RMSUSLS T87 T87 T87 T87
 ERASELPC
 LMSUSLS M37 M37 M37
 RMSUSLS M36A M36A M37 M37 M37 M37 M37 M37
 ERASEMATCHDIFF
 LMSUSLS M23
 RMSUSLS M23 M23
 ERASEMD1
 LMSUSLS T21
 RMSUSLS T20 T21 T40
 ERASEML1
 LMSUSLS T24 T25
 RMSUSLS T20 T24 T25
 ERASEMM1
 LMSUSLS T26 T27
 RMSUSLS T22 T24 T26 T27
 ERASEMM2
 LMSUSLS T22 T23
 RMSUSLS T20 T22 T23
 ERASEMYAL
 LMSUSLS M24 M24 M24 M24
 RMSUSLS M24 M24 M24 M24 M24
 ERASEORI
 LMSUSLS T44 T46 T47 T48
 RMSUSLS T42 T46 T46 T46 T46 T47 T48
 EVALGOAL
 LMSUSLS Q1 C7 C7 C3 C4 C8
 RMSUSLS Q1 C1 C2 C2 C3 C4 C8 C10 C11 C36 M26 M27 M47 M51
 EXTNAME1
 LMSUSLS F51 F51 F57
 NESTEDL F51
 RMSUSLS F50 F51 F57
 EXTNAME12
 LMSUSLS F53
 RMSUSLS F52 F53
 EXTNAME1
 LMSUSLS F2

RMSUSLS F1 F2
 EXTNAME12
 LMSUSLS F3
 RMSUSLS F2 F3
 EXTNAME1
 LMSUSLS F30
 RMSUSLS F20 F30
 EXTNAME
 LMSUSLS X1
 RMSUSLS F10 X1
 FAIL
 LMSUSLS E20 E21
 RMSUSLS E3 E4 E20 E21 E23 M48 M48
 FAILED
 LMSUSLS E23 E23
 RMSUSLS E20 E21
 FEASASC
 LMSUSLS Q2 F51 M38 M48
 NESTEDL M40
 RMSUSLS Q2 F51 F51 M38A M38 M38A M38 M38 M48 M32
 FILEDESASC
 LMSUSLS F50 M38
 RMSUSLS F50 F51 M38A M38 M38A M38 M38
 FILEGOAL
 LMSUSLS F8
 RMSUSLS Q1 E10 E11 E36 F8 M26 M27 M47 M51
 FILEDCPPROG
 LMSUSLS Q02 F1
 RMSUSLS Q08 Q07 Q07 Q07 Q07 Q02 Q02 F1 M9
 FILEOBJECT
 LMSUSLS F10
 RMSUSLS Q1 F10 M41
 FORM2IMPIMETHOD
 LMSUSLS M5
 FORMOFMETHOD
 LMSUSLS M4
 GENDESASC
 LMSUSLS M35 M35
 RMSUSLS M34 M34 M34 M35 M35
 GENDESASC2
 LMSUSLS M36 M36 M37 M38 M38 M38
 NESTEDL M37
 RMSUSLS M35 M35 M36 M37 M38 M38 M38
 GETLPCOMMON
 LMSUSLS M35
 GPRINIT
 LMSUSLS E0
 LMSUSLS Q1 E0
 HASACTUALORI
 LMSUSLS E35 F7 F7 F8 F8 F8 F8 M20 M20 M25 M26 M27 M30 M36 M40 M47 M51 V1
 V3 V4 V5 V6
 NESTEDL E30 E35 F7 F7 F8 F8 F8
 RMSUSLS Q1 E10 E11 E36 M26 M27 M47 M51
 HASALTOTIFFR
 LMSUSLS E29
 NESTEDL E26
 RMSUSLS E25 M25
 HASANTEC
 LMSUSLS E3 E20 V3 V5
 NESTEDL E4 E21 V1 V4
 RMSUSLS E10 E11
 HASDESASC
 LMSUSLS F8 F8 F51 M38 M48 M51 M52 V4 V5
 NESTEDL F8 M48
 RMSUSLS E11 F51 F51 M36 M36A M36A M36A M38 M47
 HASDESIMPORI
 LMSUSLS E36 F7 F7 M20 M20 M25 M26 M27 V1 V3
 NESTEDL F7 F7
 RMSUSLS Q1 E10 E36
 HASDITIFIC
 LMSUSLS E2 E29 E3 E4 E8 E10 E30 M27 M47 M50 M51 M57 V4 V5 V6
 NESTEDL E1 E30 M26
 RMSUSLS E29 E8 E10 E11 M26 M27 M47 M51 M57 M57
 HASDITIFFR
 LMSUSLS F8 F8 M30 V6
 NESTEDL F8 F8
 RMSUSLS M26 M27 M51
 HASEXTREPR
 LMSUSLS F44 V2 V4
 RMSUSLS F5 F44 M6
 HASLMS
 LMSUSLS F30 F34 F35 F40
 RMSUSLS F13 F34 F35 F40

HASL INK
 LMSUSLS Q2 F1 F3 F4 F26 F30 F32
 NESTEDL Q08 Q09 Q07 Q01 Q012 Q2 Q2 F1 F3 F4 F5 F27 F30 F32 F34 F35
 RMSUSLS Q08 Q09 Q07 Q01 Q012 F1 F3 F4 F26 F30 F32 F34 F35 K11
 HASL P-COMPO
 LMSUSLS M35 M36 M37 M38 M39
 NESTEDL M35 M36
 RMSUSLS F3 F4 M36A M37
 HASL MOV-COMPO
 LMSUSLS M34 M36A M36M M36N M36R
 NESTEDL M36A M36R
 RMSUSLS Q1
 HASL NAME
 LMSUSLS Q2 Q2 K10
 RMSUSLS Q2 Q2 F5 K10
 HASL WFEAS
 LMSUSLS M40 M40N M40R M40S M40T
 NESTEDL M40 M40N M40S M40T M40S M40T
 RMSUSLS Q03 Q04 M40
 HASL WFEASORD
 LMSUSLS Q03 Q04
 RMSUSLS Q03 Q04 Q03 Q04
 HASL OP
 LMSUSLS F8 F8N M3 M4 M5 M57 V4 V5
 NESTEDL F8
 RMSUSLS F11 M31 M37 M47
 HASL OFFER
 LMSUSLS F87 M10
 NESTEDL F87 F87 M12
 RMSUSLS M47
 HASL OFFERASG
 LMSUSLS M43 M43R M45 M45I
 NESTEDL M43 M45 M45I M46 M46T
 RMSUSLS M43 M43 M43F M43L M45 M45I M45I
 HASL REP
 LMSUSLS X3 X4 X5 X6
 RMSUSLS X1 X2 X3 X4 X5 X5 X6
 HASL SUPERGOAL
 LMSUSLS E2 E4 E10 E11 E12 E21 E30 E36 V1 V3 V4 V5 V6
 NESTEDL E30 E36
 RMSUSLS Q1 E10 E11 E36 M26 M27 M47 M51
 HASL TOPNODE
 LMSUSLS Q4 Q08 Q09 Q07 Q01 Q012 F44 K0 K9 K11 T1 T2 T6 T10 T11 T20 T23 T30
 T33 T40 T43 C1 X1 X6
 NESTEDL F44 K0 C4
 RMSUSLS Q1 E0 F44 C1
 HASL TRACE LEVEL
 LMSUSLS E2 E2R E3 E4 E8 E10 E11 E12 E20 E21 E22 E24 E25 E26 E31 V1 V3 V4 V5
 V6
 RMSUSLS E0 E2R E3 V1 V3 V4 V5 V6
 HASL VAL
 LMSUSLS Q4 Q08 Q09 Q07 Q01 Q012 F47 K3 K6 K7 T13 T22 T23 T32 T33 T42 T43 C3
 X3
 NESTEDL F46 K6 K7 C2 X4
 RMSUSLS Q1 F47 T5 T6 T13 T22 T22 T23 T23 T32 T33 T33 T42 T42 T43
 T43 C3
 HASL VAR
 LMSUSLS M36A M36 M36A M36 M36F
 NESTEDL M36R M36
 RMSUSLS Q1
 HASL VAR INK
 LMSUSLS M38 M38F
 NESTEDL M38
 RMSUSLS Q1
 INCR INK
 LMSUSLS T21 T22
 RMSUSLS T20 T21 T21 T22
 INCR INK
 LMSUSLS T20 T21 T22 T23
 NESTEDL T21
 RMSUSLS Q4 T20 T21 T22 T23
 INCR I
 LMSUSLS M32 M32S
 IS2 IMPUT
 LMSUSLS M40 M5
 IS2 APPLY GOAL
 LMSUSLS F8 F8N M3 M4 M5 M57 V4 V5
 NESTEDL F30 F8
 RMSUSLS F11 M47
 IS2 DESCRIBE DORI
 LMSUSLS M75
 IS2 DORI
 LMSUSLS F13 F30 F35
 RMSUSLS E0
 IS2 ORMDP
 LMSUSLS M4 M5 M33 M33S
 IS2 ORMDP
 LMSUSLS M3 M31 M32
 RMSUSLS Q1
 IS2 ORMDP GOAL
 LMSUSLS E2 E2R E4 E30 F8 F8N M2 M47 V6
 NESTEDL E30 F8 F8N
 RMSUSLS M26 M27 M51
 IS2 ORMDP
 LMSUSLS F18
 NESTEDL F18
 RMSUSLS F18
 IS2 ORMDP
 LMSUSLS F51
 NESTEDL F51
 RMSUSLS F51
 IS2 ORMDP
 LMSUSLS F17 F81
 NESTEDL F17
 RMSUSLS F40
 IS2 ORMDP GOAL
 LMSUSLS E2R E8
 NESTEDL E11 E2
 RMSUSLS F74 F85 F81 F8N
 IS2 SET
 LMSUSLS M41 M37 M33 M33S
 IS2 TRANSFORM GOAL
 LMSUSLS E22 E24 E25 E26 F35 F77 F7N M1 V1 V3
 NESTEDL E30 E30 E35 F7 F7N
 RMSUSLS Q1 E10 E36
 LAST ORMDP
 LMSUSLS F56 F57
 RMSUSLS E0 F56 F56 F57 F57
 LAST ORMDP
 LMSUSLS F5 F5E
 RMSUSLS E0 F5 F5 F5E F5E
 LAST ORMDP
 LMSUSLS F13 F19 F34
 NESTEDL F34
 RMSUSLS E0 F13 F13 F19 F19 F34 F34
 LINKS
 LMSUSLS Q4 Q08 Q09 Q07 Q01 Q012 F46 F47 K1 K4 K5 K6 K9 K11 T2 T6 T10 T11 T12
 T13 T14 T20 T21 T22 T23 T30 T31 T32 T33 T40 T41 T42 T43 C2 C3 X1 X2
 NESTEDL F46 K4 K5 T1 T3 C4 X4
 RMSUSLS Q1 F46 F47 T1 T3 T5 T6 T10 T12 T13 C2 C3
 LOCENTR
 LMSUSLS F24 F42L K8 K9 K11
 NESTEDL F25 F42
 RMSUSLS F24 F42L K3 K6 K7 K8 K9 K9 K11
 LOC ORMDP RESULT
 LMSUSLS M31 M37 M33 M33S D2
 RMSUSLS M31 M32 M33 M33S D2
 MATCH ORMDP
 LMSUSLS F28 K11
 NESTEDL K9
 RMSUSLS F13 F15 F17 F21 F28
 MATCH ORMDP
 LMSUSLS F21 F470 M23E K0 K1 K3 K4 K5 K6 K7
 NESTEDL F28 F47
 RMSUSLS F13 F15 F17 F21 F470 M20 M23E K0 K0 K1 K3 K4 K5 K6 K7
 MATCH RESI
 LMSUSLS F20 F22 F47N
 NESTEDL F20 F23 F40 F42 F470 F42L
 RMSUSLS F20 F22 F47N K11
 MATCH ORMDP STXAM
 LMSUSLS F20 F40
 RMSUSLS F13 F15 F17 F20 F40
 MATCH ORMDP STR
 NESTEDL K1
 RMSUSLS Q1
 MATCH ORMDP SUBT
 LMSUSLS M21
 RMSUSLS M21 K10
 MATCH VAL
 LMSUSLS M24 M24E M24N M24S
 NESTEDL M24 M24E M24N M25
 RMSUSLS E25 M22 M24 M24E M24N M24S
 MATCH VAL
 LMSUSLS M22
 RMSUSLS M21 M22
 MC(N)I
 LMSUSLS Q1

```

METHODSEXKH
LMSUSLS F22 F23 F30
NESTEDL F30
RMSUSLS F3 F4 M4R M4T
MORE OA
LMSUSLS F9 M F9 IN
RMSUSLS F9 I F9 IN F9 IN
MOVEOP METHOD
LMSUSLS M50 M52 M5R M59
RMSUSLS M3 M50 M52 M5R M59
NEXT GOAL APPLY
LMSUSLS E11
NESTEDL E12
RMSUSLS M51
NEXT GOAL TRANS
LMSUSLS E10
NESTEDL E12
RMSUSLS M2R M27
ONE T SUCC
LMSUSLS F42 F470 F471 F47R F47S
RMSUSLS F41 F42 F470 F471 F47R F47S F47U
ONE T SUCC H
LMSUSLS F47U
RMSUSLS F470 F471 F47R F47S F47U
RECOG GOAL
LMSUSLS I2 E7R E3 F4 F7 F7N F8 F8N F9 F9N
RMSUSLS F6 F7 F7N F8 F8N F9 F9N
REDUCE METHOD
LMSUSLS M30 M4R M42 M49N
RMSUSLS M2 M30 M4R M42 M49N
REDUCE OPCH
RMSUSLS M33 M33S
REMLAST
LMSUSLS T12 T13 T14
RMSUSLS T10 T11 T12 T12 T13 T14 T14
REMLIN
LMSUSLS T10 T11 T12 T13 T14
NESTEDL T10 T12
RMSUSLS QA T10 T10 T11 T12 T12 T13 T14
REPLISP
RMSUSLS F36 F3R
REPRISP
RMSUSLS F34 F3S
RESULT SETUP
LMSUSLS K10
RMSUSLS K9 K10
RETRY
LMSUSLS M30 M4R M49 M49N M50 M52 M5R M59
RMSUSLS E27 E3 M4R M49 M49N M5R M59
RETRY TRANS
LMSUSLS E29 E26 M24S
NESTEDL E24 M24N
SELECT OES ASG
LMSUSLS M30 M36A M36M M36N M36R M37S
NESTEDL M37
RMSUSLS M31 M37 M34 M36A M36M M36N M36R M37S
SELECT METHOD
LMSUSLS M1 M2 M3 M4 M5
RMSUSLS F1 F27 F31 M1 M2 M3 M4 M5
SELECT METHOD WOU
LMSUSLS E35
NESTEDL E30
SELECT OP
LMSUSLS M31 M37 M33 M33S
RMSUSLS M30 M31 M37 M33 M33S
SPLIT OR
LMSUSLS F36 F3R
RMSUSLS F34 F3S F36 F3R
SPROLL #10 APP
LMSUSLS M51
RMSUSLS M50 M51 M57
SPROLL #10 TRANS
LMSUSLS M2R M27
RMSUSLS M24 M2R M27
SUCCEED
LMSUSLS E10 E11 E12 F41
RMSUSLS E10 E11 E12 E12 F41 F47 M20S M25 M41
SUCCEED D
LMSUSLS E10 E11 E12
TEST ONE 1
RMSUSLS F10 F11
TEST ONE 1 F
LMSUSLS F11

```

```

RHSUSLS F10 F11
TESTONE10
LHSUSLS F13 F14
RHSUSLS F11 F13 F14
TESTONE15
LHSUSLS F15 F17 F420
NESTEDL F42
RHSUSLS F14 F15 F17 F420
TRACEASG
LHSUSLS V7 V8 V9
RHSUSLS V4 V5 -V7 -V8 -V9
TRACEGOAL
LHSUSLS V1 V3 V4 V5 V8
RHSUSLS F6 -V1 -V3 -V4 -V5 -V8
TRACEIND
LHSUSLS QA E23R E31 E36 F5 F40 V1 V2 V3 V4 V5 V6 V7 V8 V9 X40
RHSUSLS E0 E31 -E31 E36 -E36 V1 -V1 V3 -V3 V4 -V4 V5 -V5 V6 -V6
TRACEOBJ
LHSUSLS V2
RHSUSLS V1 -V2 V3 V4 V5 V6
TRACING
RHSUSLS QA E2 E2R E3 E4 E8 E10 E11 E12 E20 E21 E22 E23R E24 E25 E26 E31 E36
F5 F40 V1 V2 V3 V4 V5 V6 V7 V8 V9
TRANSJ2
LHSUSLS M23
RHSUSLS M20 -M23
TRANSJ3
LHSUSLS M24 M25
RHSUSLS E25 M23 M24 -M25
TRANSJFORMACTION
LHSUSLS M70 M70S
RHSUSLS M1 -M70 -M70S
TRYDIAGNALS
LHSUSLS E30 E35
RHSUSLS E2R E8 E23R E24 E26 -E30 -E35
TRYAPP
LHSUSLS M40 M40H M40R M40T
NESTEDL M40 M40H M40S
RHSUSLS M34 M34A M34H M34H -M40 M40H -M40H -M40R M40L M42 M44 -M45 -M46T M48
M57 M58
TRYAPP2
LHSUSLS M45 M45J M46 M46T
RHSUSLS -M41 M43 M43F M43L M44 -M45 -M46 -M46T M48 M50
TRYAPPDIFFRESTIP
LHSUSLS M43 M43F M43L
NESTEDL M43 M43F
RHSUSLS M42 M43 -M43F -M43L
TRYAPPRESLAT
LHSUSLS M47 M57
RHSUSLS M45 -M47 -M57
TRYAPPW
LHSUSLS M40U
RHSUSLS M40H M40R -M40U
VARDOMAIN
LHSUSLS Q6 Q6 F M34A M36 M36A M36
NESTEDL M340
RHSUSLS Q1
XRCOLL
LHSUSLS X2 X3 X4 X5
NESTEDL X5 X6
RHSUSLS X1 X7 -X3 -X6 -X5

```

Appendix D. SAMPLE CREATION PRODUCTIONS

DISPLAY OF NET PHONES FOR MRO

```

DA-1
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE DUMMY)))
(SATISFIES X4 (EQ X4 (QUOTE BANANAS)))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-1)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-2
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE DUMMY)))
(SATISFIES X4 (EQ X4 (QUOTE BANANAS)))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-2)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-4
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE WALKTO)))
(SATISFIES X4 (EQ X4 (QUOTE PLACE2)))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-4)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-3
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE MOVE:TO)))
(SATISFIES X4 (EQ X4 (QUOTE PLACE2)))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-3)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-10
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE WALKTO)))
(SATISFIES X4 (EQ X4 (QUOTE (MOVE: BANANAS))))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-10)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-9
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE MOVE:TO)))
(SATISFIES X4 (EQ X4 (QUOTE (MOVE: BANANAS))))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-9)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
DA-15
LMS (ASSIGN DA X3 X4) (SATISFIES X3 (EQ X3 (QUOTE DUMMY)))
(SATISFIES X4 (EQ X4 (QUOTE ONPROX)))
(NOT
  ((EXISTS X1 X2) (ASSIGN DA X1 X2)
    (NOT ((EXISTS) (VEQ X1 X3) (VEQ X2 X4))))))
RHS (ISSAME DA DA (QUOTE DA-15)) (NOT (ASSIGN DA X3 X4))
VARS X4 X3 DA
.....
LC-1
LMS (GETLPCOMPON L) (SATISFIES L (EQ L (QUOTE (P-1))))
RHS (HASLPCOMPON (QUOTE (P-1)) (QUOTE MONREYPLACE)) (NOT (GETLPCOMPON L))
VARS L
LN-1
LMS (HASL INK DO (O NO)) (SATISFIES L (EQ L (QUOTE MONREYPLACE)))
(SATISFIES NO (EQ NO O))
RHS (HASNAME DO (QUOTE (P-1)) (HASLPCOMPON (QUOTE (P-1)) (QUOTE MONREYPLACE)))
(NOT (EXISTS (P-1) (NOT (HASL INK DO (O NO))))
VARS NO (O NO)
LA-1
LMS (APPLYLOCPRG L O) (SATISFIES L (EQ L (QUOTE (P-1))) (HASTOPNOE O N1))
(L INK L1 N1 NO) (SATISFIES L1 (EQ L1 (QUOTE MONREYPLACE))) (HASVAL NO V)
RHS (LOCPRGRESLT L O V) (NOT (APPLYLOCPRG L O))
VARS V NO L1 N1 O L
LC-2

```

```

LMS (GETLPCOMPON L) (SATISFIES L (EQ L (QUOTE (P-2)))
RHS (HASLPCOMPON (QUOTE (P-2)) (QUOTE MONREYPLACE)) (NOT (GETLPCOMPON L))
VARS L
LN-2
LMS (HASL INK DO (O NO)) (SATISFIES L (EQ L (QUOTE MONREYPLACE)))
(SATISFIES NO (EQ NO O))
RHS (HASNAME DO (QUOTE (P-2)) (HASLPCOMPON (QUOTE (P-2)) (QUOTE MONREYPLACE)))
(NOT (EXISTS (P-2) (NOT (HASL INK DO (O NO))))
VARS NO (O NO)
LA-2
LMS (APPLYLOCPRG L O) (SATISFIES L (EQ L (QUOTE (P-2))) (HASTOPNOE O N1))
(L INK L1 N1 NO) (SATISFIES L1 (EQ L1 (QUOTE MONREYPLACE))) (HASVAL NO V)
RHS (LOCPRGRESLT L O V) (NOT (APPLYLOCPRG L O))
VARS V NO L1 N1 O L
LC-3
LMS (GETLPCOMPON L) (SATISFIES L (EQ L (QUOTE (P-3)))
RHS (HASLPCOMPON (QUOTE (P-3)) (QUOTE BOXPLACE)) (NOT (GETLPCOMPON L))
VARS L
LN-3
LMS (HASL INK DO (O NO)) (SATISFIES L (EQ L (QUOTE BOXPLACE)))
(SATISFIES NO (EQ NO O))
RHS (HASNAME DO (QUOTE (P-3)) (HASLPCOMPON (QUOTE (P-3)) (QUOTE BOXPLACE)))
(NOT (EXISTS (P-3) (NOT (HASL INK DO (O NO))))
VARS NO (O NO)
LA-3
LMS (APPLYLOCPRG L O) (SATISFIES L (EQ L (QUOTE (P-3))) (HASTOPNOE O N1))
(L INK L1 N1 NO) (SATISFIES L1 (EQ L1 (QUOTE BOXPLACE))) (HASVAL NO V)
RHS (LOCPRGRESLT L O V) (NOT (APPLYLOCPRG L O))
VARS V NO L1 N1 O L
.....
OB-1 "OB"
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)
(SATISFIES X3 (EQ X3 (QUOTE BOXPLACE))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE PLACE2))) (L INK X6 X2 X7)
(SATISFIES X6 (EQ X6 (QUOTE MONREYPLACE))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE PLACE1)))
RHS (ISSAME X1 (QUOTE (INITIAL OBJECT) (QUOTE OB-1)))
(HASLMS (QUOTE OB-1) (CET (QUOTE OB-1)) (QUOTE LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
OB-2
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)
(SATISFIES X3 (EQ X3 (QUOTE BOXPLACE))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE PLACE2))) (L INK X6 X2 X7)
(SATISFIES X6 (EQ X6 (QUOTE MONREYPLACE))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE PLACE2)))
RHS (ISSAME X1 (QUOTE O-1) (QUOTE OB-2))
(HASLMS (QUOTE OB-2) (CET (QUOTE OB-2) (QUOTE LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
OB-3 "OB"
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)
(SATISFIES X3 (EQ X3 (QUOTE BOXPLACE))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE PLACE2))) (L INK X6 X2 X7)
(SATISFIES X6 (EQ X6 (QUOTE MONREYPLACE))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE ONPROX)))
RHS (ISSAME X1 (QUOTE O-2) (QUOTE OB-3))
(HASLMS (QUOTE OB-3) (CET (QUOTE OB-3) (QUOTE LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
OB-4
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)
(SATISFIES X3 (EQ X3 (QUOTE BOXPLACE))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE PLACE2))) (L INK X6 X2 X7)
(SATISFIES X6 (EQ X6 (QUOTE MONREYPLACE))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE PLACE2)))
RHS (ISSAME X1 (QUOTE O-3) (QUOTE OB-4))
(HASLMS (QUOTE OB-4) (CET (QUOTE OB-4) (QUOTE LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
OB-5 "OB"
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)
(SATISFIES X3 (EQ X3 (QUOTE BOXPLACE))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE PLACE2))) (L INK X6 X2 X7)
(SATISFIES X6 (EQ X6 (QUOTE MONREYPLACE))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE (MOVE: BANANAS))))
RHS (ISSAME X1 (QUOTE O-5) (QUOTE OB-5))
(HASLMS (QUOTE OB-5) (CET (QUOTE OB-5) (QUOTE LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
OB-6
LMS (TESTONE T X1) (HASTOPNOE X1 X2) (L INK X3 X2 X4)

```

```

(SATISFIES X3 (EQ X3 (QUOTE (BOXPLAC2))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE (PLAC2))) (LINKS X6 X7 X7)
(SATISFIES X6 (EQ X6 (QUOTE (MONKEYPLAC2))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE (PLAC2)))
RHS (ISSAME X1 (QUOTE (0-6)) (QUOTE (0-6))
(HASLMS (QUOTE (0-6)) 8 (GET (QUOTE (0-6)) (QUOTE (LMS)))
VARS X7 X6 X5 X4 X3 X2 X1
08-7 "OR"
LMS (TESTONET X1) (HASTOPNODE X1 X2) (LINKS X3 X7 X4)
(SATISFIES X3 (EQ X3 (QUOTE (BOXPLAC2))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE (UNDERBANANAS))) (LINKS X6 X7 X7)
(SATISFIES X6 (EQ X6 (QUOTE (MONKEYPLAC2))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE (UNDERBANANAS)))
RHS (ISSAME X1 (QUOTE (0-7)) (QUOTE (0-7))
(HASLMS (QUOTE (0-7)) 8 (GET (QUOTE (0-7)) (QUOTE (LMS)))
VARS X8 X7 X6 X5 X4 X3 X2 X1
08-8
LMS (TESTONET X1) (HASTOPNODE X1 X2) (LINKS X3 X7 X4)
(SATISFIES X3 (EQ X3 (QUOTE (BOXPLAC2))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE (UNDERBANANAS))) (LINKS X6 X7 X7)
(SATISFIES X6 (EQ X6 (QUOTE (MONKEYPLAC2))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE (ONBOX)))
(NOT
((EXISTS X9 X10 X11) (LINKS X9 X7 X10)
(SATISFIES X9 (EQ X9 (QUOTE (MONKEYHAND))) (HASVAL X10 X11)
(SATISFIES X11 (EQ X11 (QUOTE (BANANAS))))
RHS (ISSAME X1 (QUOTE (0-8)) (QUOTE (0-8))
(HASLMS (QUOTE (0-8)) 11 (GET (QUOTE (0-8)) (QUOTE (LMS)))
VARS X5 X7 X6 X5 X4 X3 X2 X1
08-9
LMS (TESTONET X1) (HASTOPNODE X1 X2) (LINKS X3 X7 X4)
(SATISFIES X3 (EQ X3 (QUOTE (BOXPLAC2))) (HASVAL X4 X5)
(SATISFIES X5 (EQ X5 (QUOTE (UNDERBANANAS))) (LINKS X6 X7 X7)
(SATISFIES X6 (EQ X6 (QUOTE (MONKEYPLAC2))) (HASVAL X7 X8)
(SATISFIES X8 (EQ X8 (QUOTE (ONBOX))) (LINKS X9 X7 X10)
(SATISFIES X9 (EQ X9 (QUOTE (MONKEYHAND))) (HASVAL X10 X11)
(SATISFIES X11 (EQ X11 (QUOTE (BANANAS)))
RHS (ISSAME X1 (QUOTE (0-9)) (QUOTE (0-9))
(HASLMS (QUOTE (0-9)) 11 (GET (QUOTE (0-9)) (QUOTE (LMS)))
VARS X11 X10 X9 X8 X7 X6 X5 X4 X3 X2 X1

```

KREF OF NET PRODS FROM MRO RUN

```

APPLYLOCPOD
LMSUSES LA-1 LA-7 LA-3
RHSUSES LA-1 LA-2 LA-3
ASSIGNSN
LMSUSES DA-1 DA-2 DA-4 DA-3 DA-10 DA-9 DA-15
NESTEDL DA-1 DA-2 DA-4 DA-3 DA-10 DA-9 DA-15
RHSUSES DA-1 DA-2 DA-4 DA-3 DA-10 DA-9 DA-15
GETLPCOMPON
LMSUSES LC-1 LC-7 LC-3
RHSUSES LC-1 LC-2 LC-3
HASLTMK
LMSUSES LN-1 LN-7 LN-3
RHSUSES LN-1 LN-2 LN-3
HASTOPNODE
LMSUSES LA-1 LA-7 LA-3 OR-1 OR-7 OR-3 OR-4 OR-5 OR-6 OR-7 OR-8 OR-9
HASVAL
NESTEDL OR-8
LINKS
NESTEDL OR-8
TESTONET
LMSUSES OR-1 OR-7 OR-3 OR-4 OR-5 OR-6 OR-7 OR-8 OR-9

```

Appendix E. DETAILED DESCRIPTION ON THE MONKEY AND BANANAS TASK

MONKEY WITH EPMSY AND HENELL TABLE OF CONNECTIONS

```

G-1 : TRANSFORM INITIAL:OBJECT TO DESIRED:OBJECT (FROM TOP)
LOC:PPROG LP-1 (MONKEY:HAND)
G-2 : REDUCE UNDER TO BANANAS AT (MONKEY:HAND) OF INITIAL:OBJECT (DIFFIC 305)
(FROM G-1)
LOC:PPROG LP-2 (MONKEY:PLACE)
LOC:PPROG LP-3 (BOX:PLACE)
G-3 : APPLY CLIMB TO INITIAL:OBJECT (DIFFIC 100) (FROM G-2)
ASSIGNS DUMMY - BANANAS
G-4 : REDUCE PLACE1 TO PLACE2 AT (MONKEY:PLACE) OF INITIAL:OBJECT (DIFFIC
100) (FROM G-3)
APPLY WALK TO INITIAL:OBJECT GET 0-1 (WALK TO PLACE2)
G-4 SUCCEEDS
G-5 : APPLY CLIMB TO 0-1 (DIFFIC 100) (FROM G-3 AND G-4)
ASSIGNS DUMMY - BANANAS
0-1 (BOX:PLACE PLACE2 MONKEY:PLACE PLACE2)
APPLY CLIMB TO 0-1 GET 0-2
G-5 SUCCEEDS
G-3 SUCCEEDS
G-2 SUCCEEDS
G-6 : TRANSFORM 0-2 TO DESIRED:OBJECT (FROM G-1 AND G-2)
0-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON:BOX)
G-7 : REDUCE UNDER TO BANANAS AT (MONKEY:HAND) OF 0-2 (DIFFIC 305) (FROM
G-6)
0-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON:BOX)
G-8 : APPLY CLIMB TO 0-2 (DIFFIC 100) (FROM G-7)
ASSIGNS DUMMY - BANANAS
0-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON:BOX)
G-9 : REDUCE UNDER TO PLACE2 AT (MONKEY:PLACE) OF 0-2 (DIFFIC 100) (
FROM G-8)
0-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON:BOX)
APPLY WALK TO 0-2 GET 0-3 (WALK TO PLACE2)
0-3 IS THE SAME AS 0-1
G-9 SUCCEEDS
G-10 : APPLY CLIMB TO 0-1 (DIFFIC 100) (FROM G-8 AND G-9)
ASSIGNS DUMMY - BANANAS
0-1 (BOX:PLACE PLACE2 MONKEY:PLACE PLACE2)
APPLY CLIMB TO 0-1 GET 0-4
0-4 IS THE SAME AS 0-2
G-10 SUCCEEDS
G-8 SUCCEEDS
G-7 SUCCEEDS
G-11 : TRANSFORM 0-2 TO DESIRED:OBJECT (FROM G-6 AND G-7)
0-2 (BOX:PLACE PLACE2 MONKEY:PLACE ON:BOX)
REPEATED GOAL: G-11 G-6
RETRYING OLD G-4
G-4 FAILED
RETRYING G-3
G-3 FAILED
RETRYING G-2
G-12 : APPLY GET BANANAS TO INITIAL:OBJECT (DIFFIC 200) (FROM G-2)
ASSIGNS DUMMY - BANANAS
G-13 : REDUCE PLACE2 TO UNDER:BANANAS AT (BOX:PLACE) OF INITIAL:OBJECT (
DIFFIC 200) (FROM G-12)
APPLY WALK TO INITIAL:OBJECT GET 0-5 (WALK TO UNDER:BANANAS)
G-13 SUCCEEDS
G-14 : APPLY GET BANANAS TO 0-5 (DIFFIC 200) (FROM G-12 AND G-13)
ASSIGNS DUMMY - BANANAS
0-5 (BOX:PLACE PLACE2 MONKEY:PLACE UNDER:BANANAS)
G-15 : REDUCE PLACE2 TO UNDER:BANANAS AT (BOX:PLACE) OF 0-5 (DIFFIC 200)
(FROM G-14)
0-5 (BOX:PLACE PLACE2 MONKEY:PLACE UNDER:BANANAS)
G-16 : APPLY MOVE BOX TO 0-5 (DIFFIC 100) (FROM G-15)
ASSIGNS MOVE:10 - UNDER:BANANAS
0-5 (BOX:PLACE PLACE2 MONKEY:PLACE UNDER:BANANAS)
G-17 : REDUCE UNDER:BANANAS TO PLACE2 AT (MONKEY:PLACE) OF 0-5 (
DIFFIC 100) (FROM G-16)
0-5 (BOX:PLACE PLACE2 MONKEY:PLACE UNDER:BANANAS)
APPLY WALK TO 0-5 GET 0-6 (WALK TO PLACE2)
0-6 IS THE SAME AS 0-1
0-6 IS THE SAME AS 0-1
G-17 SUCCEEDS
G-18 : APPLY MOVE BOX TO 0-1 (DIFFIC 100) (FROM G-16 AND G-17)
ASSIGNS MOVE:10 - UNDER:BANANAS
0-1 (BOX:PLACE PLACE2 MONKEY:PLACE PLACE2)
APPLY MOVE BOX TO 0-1 GET 0-7 (MOVE TO UNDER:BANANAS)
G-18 SUCCEEDS
G-16 SUCCEEDS

```


(CONTROL FLOW FOR FOR MONEY WITH ERNST AND NEWMILL TABLE OF CONNECTIONS)

ENKFVOCIALD

OT-1		O	1.
OS-1	E		1.
F6-1		F	1.
V1-1		V	1.
F7-1		F	2..
X1-1		X	5.....
F11-1		F	2..
K0-1		K	7.....
F20-1		F	8.....
E1-1	E		1.
M1-1	H		2..
OE-1		O	1.
F1-1		F	4....
K1A-1		K	1.
M2-1	H		1.
D1-1		D	1.
LA-1-1		L	1.
O2-1		O	2..
M2-1	H		6.....
F6-2		F	1.
V6-1		V	1.
F9-1		F	1.
E1-2	E		1.
M2-1	H		2..
LA-1-2		L	1.
M32-1	H		15.....
F50-1		F	9.....
OF-1		O	2..
M4M4-1	H		2..
GOC-1		O	1.
F1-2		F	4....
OE-1		O	1.
M42-1	H		1.
D1-2		D	1.
LA-2-1		L	1.
D2-2		D	2..
M43F-1	H		3...
GOG1-1		O	1.
F1-3		F	4....
OE-2		O	1.
M47-2	H		1.
D1-3		D	1.
LA-3-1		L	1.
D2-3		D	2..
M43F-2	H		3...
F6-3		F	1.
V4-1		V	2..
F8-1		F	1.
E1-3	E		1.
M3-1	H		2..
F6-4		F	1.
V6-2		V	1.
F9-2		F	1.
E1-4	E		1.
M2-2	H		2..
LA-2-2		L	1.
M32-5	H		12.....
F50-3		F	9.....
OF-3		O	2..
M4M4-2	H		6.....
OMU-1		O	1.
C1-1		C	3...
T43-1		T	1.
M41-1	H		1.
F10-2		F	1.
X1-3		X	5.....
OB-1-1		O	1.
F11-2		F	3...
FA-2		K	5.....
F20-2		F	10.....
E11-1	E		1.
F6-5		F	1.
V5-1		V	3...
F8-2		F	1.
E1-5	E		1.
M3-2	H		2..
OF-5		O	1.
M40-5	H		1.

QAC-1				1.
C1-2			C	3...
T43-2			T	1.
M1-2	H			1.
F10-3		F		1.
X1-5		F	X	5.....
F11-3		F		2..
K0-3		K		7.....
F20-3		F		9.....
E12-1	E			9...
F6-6		F		1.
V3-1		F	V	2..
F7-2		F		1.
E1-6	E			1.
M1-2	H			2..
QK-2			O	1.
F1-4		F		1.
LN-1-1			L	1.
K10-2	H	K		1.
M21-2	H			1.
D1-4			D	1.
LA-1-3			L	1.
D2-4			D	2..
M2-2	H			6.....
F6-7		F		1.
U6-3		F	V	2..
F9-3		F		1.
E1-7	E			1.
M2-3	H			2..
LA-1-4			L	1.
M32-9	H			13.....
F50-5		F		2..
DA-2-1			D	2..
F51-1		F		3...
QF-6			O	2..
M10M-3	H			2..
QOC-2			O	1.
F1-5		F		1.
LN-2-1			L	1.
QE-3			O	1.
M12-3	H			1.
D1-5			D	1.
LA-2-3			L	1.
D2-5			D	2..
M13F-3	H			3...
QOG1-2			O	1.
F1-6		F		1.
LN-3-1			L	1.
QE-4			O	1.
M12-4	H			1.
D1-6			D	1.
LA-3-2			L	1.
D2-6			D	2..
M13F-4	H			3...
F6-8		F		1.
U4-2		F	V	3...
F6-3		F		1.
E1-8	E			1.
M3-3	H			3...
F6-9		F		1.
U6-4		F	V	2..
F9-4		F		1.
E1-9	E			1.
M2-4	H			2..
LA-2-4			L	1.
M32-13	H			10.....
F50-7		F		2..
DA-3-1			D	1.
F51-3		F		3...
DA-4-1			D	1.
F51-4		F		2..
QF-8			O	2..
M10M-4	H			6.....
QAM-2			O	1.
C1-3			C	3...
T43-3			T	1.
M11-3	H			1.
F10-4		F		1.
X1-7		F	X	5.....
DB-3-1			D	2..
F11-4		F		5.....
K0-4	K			8.....
F20-4		F		19.....

E11-2	E			1.
F6-10		F		1.
U6-2		F	V	3...
F6-4		F		1.
E1-10	E			1.
M3-4	H			2..
QF-10			O	1.
M10-10	H			1.
QAC-2			O	1.
C1-4			C	3...
T43-4			T	1.
M11-4	H			1.
F10-5		F		1.
X1-9		F	X	5.....
DB-3-2			D	1.
F11-5		F		3...
K0-6	K			3...
F40-2		F		10.....
E12-3	E			3...
F6-11		F		1.
U3-2		F	V	2..
F7M-1		F		1.
E0-1	E			6.....
M2-5	H			4.....
E21-1	E			2..
M3-5	H			3...
E21-2	E			2..
M2-6	H			5.....
F6-12		F		1.
U4-3		F	V	2..
F0-5		F		1.
E1-11	E			1.
M3-6	H			3...
F6-13		F		1.
U6-5		F	V	1.
F9-5		F		1.
E1-12	E			1.
M2-7	H			2..
LA-3-3			L	1.
M32-17	H			11.....
F50-10		F		9.....
QF-11			O	2..
M10M-5	H			2..
QOM-1			O	1.
F1-7		F		1.
LN-2-2			L	1.
QE-5			O	1.
M12-5	H			1.
D1-7			D	1.
LA-2-5			L	1.
D2-7			D	2..
M13F-5	H			3...
QAM-3			O	1.
C1-5			C	3...
T43-5			T	1.
M11-5	H			1.
F10-6		F		1.
X1-11		F	X	5.....
F11-6		F		2..
K0-7	K			7.....
F20-5		F		9.....
E11-3	E			1.
F6-14		F		1.
U6-3		F	V	3...
F0-6		F		1.
E1-13	E			1.
M3-7	H			2..
QF-13			O	1.
M10-13	H			1.
QOG1-3			O	1.
F1-8		F		1.
LN-3-2			L	1.
QE-6			O	1.
M12-6	H			1.
D1-8			D	1.
LA-3-4			L	1.
D2-8			D	2..
M13F-6	H			4.....
F6-15		F		1.
U6-6		F	V	2..
F9-6		F		1.
E1-14	E			1.
M2-8	H			2..

E.

DETAILED BEHAVIOR ON THE MONKEY AND BANANAS TASK

OPR

LA-3-5			L	1.
M32-21	H			10.....
F50-12		F		2..
DA-9-1			D	1.
F51-5		F		3...
DA-10-1			D	1.
F51-6		F		2..
OF-14			O	2..
M40-6	H			2..
ODN-2			O	1.
F1-9		F		1.
LN-2-3			L	1.
OE-7			O	1.
M47-7	H			1.
D1-9			O	1.
LA-2-6			L	1.
D2-9			O	2..
M43-7	H			7.....
F6-16		F		1.
U4-4			V	3...
F8-7		F		1.
E1-15	E			1.
M3-8	H			3...
F6-17		F		1.
U6-7			V	2..
F9-7		F		1.
E1-16	E			1.
M2-9	H			2..
LA-2-7			L	1.
M32-25	H			10.....
F50-15		F		2..
DA-4-2			D	1.
F51-7		F		3...
DA-3-2			D	1.
F51-8		F		2..
OF-16			O	2..
M40-7	H			6.....
ODN-4			O	1.
C1-6			C	3...
T43-6			T	1.
M41-6	H			1.
F10-7		F		1.
X1-13			X	6.....
DB-5-1			O	3...
F11-7		F		6.....
K0-8		K		3...
F40-3		F		10.....
K0-9		K		0.....
F20-6		F		19.....
E11-4	E			1.
F6-10		F		1.
U5-4			V	3...
F0-8		F		1.
E1-17	E			1.
M3-9	H			2..
OF-18			O	1.
M40-10	H			1.
ODN-1			O	1.
C1-7			C	3...
T43-7			T	2..
M41-7	H			1.
F10-8		F		1.
X1-15			X	5.....
F11-8		F		2..
K0-11		K		7.....
F20-7		F		9.....
E12-5	E			9...
F6-19		F		1.
U5-5			V	3...
F0-9		F		1.
E1-10	E			1.
M3-10	H			2..
OF-19			O	1.
M40-19	H			1.
ODG-1			O	1.
F1-10		F		1.
LN-2-4			L	1.
OE-8			O	1.
M42-8	H			1.
D1-10			D	1.
LA-2-8			L	1.
D2-10			O	2..
M43-8	H			4.....

F6-20		F		1.
U6-8			V	2..
F0-9		F		1.
E1-19	E			1.
M2-10	H			2..
LA-2-9			L	1.
M32-29	H			11.....
F50-18		F		4.....
OF-20			O	1.
M40-20	H			1.
ODC-3			O	1.
C1-8			C	3...
T43-9			T	1.
M41-8	H			1.
F10-9		F		1.
X1-17			X	5.....
DB-7-1			D	1.
F11-9		F		3...
U6-17			V	5.....
F20-8		F		10.....
E11-6	E			1.
F6-21		F		1.
U5-4			V	3...
F0-10		F		1.
E1-20	E			1.
M3-11	H			2..
OF-21			O	1.
M40-21	H			1.
ODC-1			O	1.
C1-9			C	3...
T6-1			T	1.
M41-9	H			1.
F10-10		F		1.
X1-19			X	7.....
DB-8-1			D	1.
F11-10		F		3...
K0-13		K		6.....
F20-9		F		11.....
E12-7	E			5.....
F6-22		F		1.
U5-3			V	2..
F7-3		F		1.
E1-21	E			1.
M1-3	H			5.....
E12-11	E			2..

PERCENTAGES OF FIRINGS OF EACH TYPE, OUT OF TOTAL 911

C 5.....
 H 27.....
 K 7.....
 F 31.....
 E 5.....
 V 5.....
 O 6.....
 C 2..
 T 1.
 D 4....
 L 2..
 D 0

DATAFLOW ANALYSIS RESULTS. FOR MONKEY WITH ERNST AND REMELL OPTION

TIME PEL. TO INSERT MAX. 910. SCALE FACTOR 18

THE FOLLOWING PREDICATES ARE IN DECP. MAX. PEL. USE ORDER

PRED'S GLOBALLY USED. MAX. PEL. TIME OF USE ABOVE DP = OVERALL-MAX / 2 ; (18)
 HASUPER.COM. ISDESCRIPED.OBJ HASPACE.LEVEL NEXT.COM. TPANS ISMOVEOP HASDIFFIC
 VAR.DOWAIN HASWAP HASMOVE.COMPON CHANGES.VAR LINKS ISSET INSET HASVAL
 HASIPMODE ASSIGAS.D ISDUPM HASRTPPO

ABOVE DP = OVERALL-MAX / 4 = 17 ; (11)

ISPRODUCE.COM HASACTUAL.OBJ HASDESIGG TRYAPP LAST.DANET HASOP.DIFFR.ASG
 TRACE.OBJ ISAPPLY.COM HASOP ISSAVE.EOV HASLP.COMPON

ABOVE DP = OVERALL-MAX / 8 = 8 ; (6)

ISTPWAP/OPM.COM HASDESIPED.OBJ NEXT.COM. APPLY LAST.DNET REPAIRP REPLM

E.

ABOVE OP = D:FULL-MAX / 16 = 4 : 131
TRACE:IND TRACING LAST:LPNET

ABOVE OP = D:FULL-MAX / 32 = 2 : 131
SUCCEED INSINS EVAL:CON

OTHERS : 199:

TPYAPP TPYAPP2 TPYAPP3 TEST:ONET ONET:SUCC MATCH:RESUME MATCH:DIFF
HASNEWFEAS FEASAG ASSIGNS XPCOLL TPYAPP:RESULT TPYAPP:DIFF:SETUP
TPY:OLD:CONLS TPY:APP:METHOD TPY:APP3 TRACE:CONL TRACE:ASC TEST:ONET
TEST:ONET TEST:ONET SUCCEEDED SPYOUT:PED:TPYAS SPYOUT:PED:APP SPLIT:OB
SELECT:OP SELECT:METHOD SELECT:DES:ASC PEPI:RESULT:SETUP PEDUCE:METHOD
PECOG:CONL ONET:SUCC MOVE:OP:METHOD MOVE:DA INIT:METHODS:EVN MATCH:SET
MATCH:VAL MATCH:RESULT MATCH:PEST MATCH:DIFF LOC:PROG:RESULT LOC:ETP
ISSAGE:CONL ISSAGE:DA ISSAGE:HASFEA HASOP:DIFF HASAGE HASLIN HASDIFF
HASANTEC GSPINIT GENDES:ASC2 GENDES:ASC FILE:OBJECT FILE:LOC:PROG FILE:CONL
FILE:DES:ASC FAILED FAIL EXTEND:EXT:ONET EXT:LPNET2 EXT:LPNET EXT:ONET2
EXT:ONET EPYSE:OBJ EPYSE:VAL EPYSE:MPI EPYSE:MPI EPYSE:MPI EPYSE:MPI
EPYSE:MATCH:DIFF EPYSE:PC EPYSE:CHOICES EPYSE:APP DIFF:EVN:RESULT
DIFF:EVN:PEST DIFF:EVN:PEST DIFF:EVN:COPY:OBJ COPY:LIN COL:ONET
COL:LPNET COL:ONET CHOISE:OLD:CONL CHOISE:SELX CHECK:PEPI ASSIGNS:IN
APPLY:RESULT APPLY:OP APPLY:LOC:PROG APPLY:DIFF:SETUP APPLY:DIFF APPLY:ON
ADDPPOD MOD:LIN

911 FIPINGS. 1777 INSTANCES (1- 1100 = 224)

Appendix F. BANANAS TASKS FOR THE MONKEY TASKS

MONKEY WITH MONY RESTRICTIVE TABLE OF CONNECTIONS

G-1 : TRANSFORM INITIAL OBJECT TO DESIRED OBJECT (FROM TOP)
LOC:PROG LP-1 (MONY:HAND)
G-2 : PEDUCE UNDER TO BANANAS AT (MONY:HAND) OF INITIAL OBJECT (DIFFIC 200)
(FROM G-1)
LOC:PROG LP-2 (MONY:PLACE)
G-3 : APPLY GET:BANANAS TO INITIAL OBJECT (DIFFIC 200) (FROM G-2)
ASSIGNS DUFFY - BANANAS
G-4 : PEDUCE PLACEZ TO UNDER:BANANAS AT (MONY:PLACE) OF INITIAL OBJECT (DIFFIC 200) (FROM G-3)
LOC:PROG LP-3 (MONY:PLACE)
G-5 : APPLY MOVE:BOX TO INITIAL OBJECT (DIFFIC 100) (FROM G-4)
ASSIGNS MOVE:TO - UNDER:BANANAS
G-6 : PEDUCE PLACEZ TO PLACEZ AT (MONY:PLACE) OF INITIAL OBJECT (DIFFIC 100) (FROM G-5)
APPLY WALK TO INITIAL OBJECT GET D-1 (WALK TO PLACEZ)
G-6 SUCCEEDS
G-7 : APPLY MOVE:BOX TO D-1 (DIFFIC 100) (FROM G-5 AND G-6)
ASSIGNS MOVE:TO - UNDER:BANANAS
D-1 (BOX:PLACE PLACEZ MONY:PLACE PLACEZ)
APPLY MOVE:BOX TO D-1 GET D-2 (MOVE:TO UNDER:BANANAS)
G-7 SUCCEEDS
G-5 SUCCEEDS
G-4 SUCCEEDS
G-3 : APPLY GET:BANANAS TO D-2 (DIFFIC 200) (FROM G-3 AND G-4)
ASSIGNS DUFFY - BANANAS
D-2 (BOX:PLACE UNDER:BANANAS MONY:PLACE UNDER:BANANAS)
G-3 : PEDUCE UNDER:BANANAS TO ON:BOX AT (MONY:PLACE) OF D-2 (DIFFIC 100) (FROM G-3)
D-2 (BOX:PLACE UNDER:BANANAS MONY:PLACE UNDER:BANANAS)
APPLY CLIMB TO D-2 GET D-3
G-3 SUCCEEDS
G-10 : APPLY GET:BANANAS TO D-3 (DIFFIC 100) (FROM G-3 AND G-9)
ASSIGNS DUFFY - BANANAS
D-3 (BOX:PLACE UNDER:BANANAS MONY:PLACE ON:BOX)
APPLY GET:BANANAS TO D-3 GET D-4
G-10 SUCCEEDS
G-8 SUCCEEDS
G-9 SUCCEEDS
G-2 SUCCEEDS
G-11 : TRANSFORM D-4 TO DESIRED OBJECT (FROM G-1 AND G-2)
D-4 (BOX:PLACE UNDER:BANANAS MONY:HAND BANANAS MONY:PLACE ON:BOX)
G-11 SUCCEEDS
G-1 SUCCEEDS

SUCCEED (TOP D-4)

RUN TIME 2 MIN. 2.23 SEC

EXAM	TRY	FIRE	WNET	E/T	E/T	T/T
1489	642	373	1200	3.94	2.29	1.72
0.0032	0.190	0.320	0.0055	SEC	AVG	

706 INSERTS 494 DELETES 123 MAPPINGS ON NEW OBJECTS
MAX ISPT LENGTH 56
COPE (FREE:FULL): (19162 : 19191) USED (6100 : 495)
PIRED 114 OUT OF 237 PRODS

TOWER OF HANOI FOUR DISK

G-1 : TRANSFORM INITIAL OBJECT TO DESIRED OBJECT (FROM TOP)
LOC:PROG LP-1 (PEG3 DISK1)
LOC:PROG LP-2 (PEG3 DISK2)
LOC:PROG LP-3 (PEG3 DISK3)
LOC:PROG LP-4 (PEG3 DISK4)
LOC:PROG LP-5 (PEG1 DISK1)
LOC:PROG LP-6 (PEG1 DISK2)
LOC:PROG LP-7 (PEG1 DISK3)
LOC:PROG LP-8 (PEG1 DISK4)
G-2 : PEDUCE UNDER TO YES AT (PEG3 DISK4) OF INITIAL OBJECT (DIFFIC 400) (FROM G-1)
LOC:PROG LP-9 (PEG2 DISK1)
LOC:PROG LP-10 (PEG2 DISK2)

LOC: PPGC LP-11 (PEG2 DISK3)

G-3 : APPLY MOVE:DISK TO INITIAL:OBJECT (DIFFIC 305) (FROM G-2)
 ASSIGNS DISK - DISK4, TO:PEG - PEG3
 G-4 : REDUCE UNDEF TO YES AT (PEG2 DISK3) OF INITIAL:OBJECT (DIFFIC 305) (FROM G-3)
 G-5 : APPLY MOVE:DISK TO INITIAL:OBJECT (DIFFIC 205) (FROM G-4)
 ASSIGNS DISK - DISK3, TO:PEG - PEG2
 G-6 : REDUCE UNDEF TO YES AT (PEG3 DISK2) OF INITIAL:OBJECT (DIFFIC 205) (FROM G-5)
 G-7 : APPLY MOVE:DISK TO INITIAL:OBJECT (DIFFIC 105) (FROM G-6)
 ASSIGNS DISK - DISK2, TO:PEG - PEG3
 G-8 : REDUCE UNDEF TO YES AT (PEG2 DISK1) OF INITIAL:OBJECT (DIFFIC 105) (FROM G-7)
 APPLY MOVE:DISK TO INITIAL:OBJECT GET G-1 (FROM:PEG PEG1 TO:PEG PEG2 DISK DISK1)
 G-8 SUCCEEDS
 G-9 : APPLY MOVE:DISK TO G-1 (DIFFIC 105) (FROM G-7 AND G-8)
 ASSIGNS DISK - DISK2, TO:PEG - PEG3
 G-1 (PEG1 DISK2 YES DISK3 YES DISK4 YES) PEG2 (DISK1 YES) PEG3 (NIL)
 APPLY MOVE:DISK TO G-1 GET G-2 (FROM:PEG PEG1 TO:PEG PEG3 DISK DISK2)
 G-9 SUCCEEDS
 G-7 SUCCEEDS
 G-6 SUCCEEDS
 G-10 : APPLY MOVE:DISK TO G-2 (DIFFIC 205) (FROM G-5 AND G-8)
 ASSIGNS DISK - DISK3, TO:PEG - PEG2
 G-2 (PEG1 DISK3 YES DISK4 YES) PEG2 (DISK1 YES) PEG3 (DISK2 YES)
 G-11 : REDUCE UNDEF TO YES AT (PEG3 DISK1) OF G-2 (DIFFIC 105) (FROM G-10)
 G-2 (PEG1 DISK3 YES DISK4 YES) PEG2 (DISK1 YES) PEG3 (DISK2 YES)
 APPLY MOVE:DISK TO G-2 GET G-3 (FROM:PEG PEG2 TO:PEG PEG3 DISK DISK1)
 G-11 SUCCEEDS
 G-12 : APPLY MOVE:DISK TO G-3 (DIFFIC 105) (FROM G-10 AND G-11)
 ASSIGNS DISK - DISK3, TO:PEG - PEG2
 G-3 (PEG1 DISK3 YES DISK4 YES) PEG2 (DISK1 YES) PEG3 (DISK2 YES)
 APPLY MOVE:DISK TO G-3 GET G-4 (FROM:PEG PEG1 TO:PEG PEG2 DISK DISK3)
 G-12 SUCCEEDS
 G-10 SUCCEEDS
 G-5 SUCCEEDS
 G-4 SUCCEEDS
 G-13 : APPLY MOVE:DISK TO G-4 (DIFFIC 305) (FROM G-3 AND G-4)
 ASSIGNS DISK - DISK4, TO:PEG - PEG3
 G-4 (PEG1 DISK4 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK2 YES)
 G-14 : REDUCE UNDEF TO YES AT (PEG2 DISK2) OF G-4 (DIFFIC 205) (FROM G-13)
 G-4 (PEG1 DISK4 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK2 YES)
 G-15 : APPLY MOVE:DISK TO G-4 (DIFFIC 105) (FROM G-14)
 ASSIGNS DISK - DISK2, TO:PEG - PEG2
 G-4 (PEG1 DISK4 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK2 YES)
 G-16 : REDUCE UNDEF TO YES AT (PEG1 DISK1) OF G-4 (DIFFIC 105) (FROM G-15)
 G-4 (PEG1 DISK4 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK2 YES)
 APPLY MOVE:DISK TO G-4 GET G-5 (FROM:PEG PEG3 TO:PEG PEG1 DISK DISK1)
 G-16 SUCCEEDS
 G-17 : APPLY MOVE:DISK TO G-5 (DIFFIC 105) (FROM G-15 AND G-16)
 ASSIGNS DISK - DISK2, TO:PEG - PEG2
 G-5 (PEG1 DISK1 YES DISK4 YES) PEG2 (DISK3 YES) PEG3 (DISK2 YES)
 APPLY MOVE:DISK TO G-5 GET G-6 (FROM:PEG PEG3 TO:PEG PEG2 DISK DISK2)
 G-17 SUCCEEDS
 G-15 SUCCEEDS
 G-14 SUCCEEDS
 G-18 : APPLY MOVE:DISK TO G-6 (DIFFIC 205) (FROM G-13 AND G-14)
 ASSIGNS DISK - DISK4, TO:PEG - PEG3
 G-6 (PEG1 DISK1 YES DISK4 YES) PEG2 (DISK3 YES DISK2 YES) PEG3 (NIL)
 G-19 : REDUCE UNDEF TO YES AT (PEG2 DISK1) OF G-6 (DIFFIC 105) (FROM G-18)
 G-6 (PEG1 DISK1 YES DISK4 YES) PEG2 (DISK3 YES DISK2 YES) PEG3 (NIL)
 APPLY MOVE:DISK TO G-6 GET G-7 (FROM:PEG PEG1 TO:PEG PEG2 DISK DISK1)
 G-19 SUCCEEDS
 G-20 : APPLY MOVE:DISK TO G-7 (DIFFIC 105) (FROM G-18 AND G-19)
 ASSIGNS DISK - DISK4, TO:PEG - PEG3
 G-7 (PEG1 DISK1 YES) PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (NIL)
 APPLY MOVE:DISK TO G-7 GET G-8 (FROM:PEG PEG1 TO:PEG PEG3 DISK DISK4)
 G-20 SUCCEEDS
 G-18 SUCCEEDS
 G-13 SUCCEEDS
 G-3 SUCCEEDS
 G-2 SUCCEEDS

G-21 : TRANSFORM G-8 TO DESIRED:OBJECT (FROM G-1 AND G-2)
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 G-22 : REDUCE UNDEF TO YES AT (PEG3 DISK3) OF G-8 (DIFFIC 305) (FROM G-21)
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 G-23 : APPLY MOVE:DISK TO G-8 (DIFFIC 205) (FROM G-22)
 ASSIGNS DISK - DISK3, TO:PEG - PEG3
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 G-24 : REDUCE UNDEF TO YES AT (PEG1 DISK2) OF G-8 (DIFFIC 205) (FROM G-23)
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 G-25 : APPLY MOVE:DISK TO G-8 (DIFFIC 105) (FROM G-24)
 ASSIGNS DISK - DISK2, TO:PEG - PEG1
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 G-26 : REDUCE UNDEF TO YES AT (PEG3 DISK1) OF G-8 (DIFFIC 105) (FROM G-25)
 G-8 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 APPLY MOVE:DISK TO G-8 GET G-9 (FROM:PEG PEG2 TO:PEG PEG3 DISK DISK1)
 G-26 SUCCEEDS
 G-27 : APPLY MOVE:DISK TO G-9 (DIFFIC 105) (FROM G-25 AND G-26)
 ASSIGNS DISK - DISK2, TO:PEG - PEG1
 G-9 (PEG1 NIL PEG2 (DISK1 YES DISK2 YES DISK3 YES) PEG3 (DISK4 YES))
 APPLY MOVE:DISK TO G-9 GET G-10 (FROM:PEG PEG2 TO:PEG PEG1 DISK DISK2)
 G-27 SUCCEEDS
 G-25 SUCCEEDS
 G-24 SUCCEEDS
 G-28 : APPLY MOVE:DISK TO G-10 (DIFFIC 205) (FROM G-23 AND G-24)
 ASSIGNS DISK - DISK3, TO:PEG - PEG3
 G-10 (PEG1 (DISK2 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK4 YES))
 G-29 : REDUCE UNDEF TO YES AT (PEG1 DISK1) OF G-10 (DIFFIC 105) (FROM G-28)
 G-10 (PEG1 (DISK2 YES) PEG2 (DISK3 YES) PEG3 (DISK1 YES DISK4 YES))
 APPLY MOVE:DISK TO G-10 GET G-11 (FROM:PEG PEG3 TO:PEG PEG1 DISK DISK1)
 G-29 SUCCEEDS
 G-30 : APPLY MOVE:DISK TO G-11 (DIFFIC 105) (FROM G-28 AND G-29)
 ASSIGNS DISK - DISK3, TO:PEG - PEG3
 G-11 (PEG1 (DISK1 YES DISK2 YES) PEG2 (DISK3 YES) PEG3 (DISK4 YES))
 APPLY MOVE:DISK TO G-11 GET G-12 (FROM:PEG PEG2 TO:PEG PEG3 DISK DISK3)
 G-30 SUCCEEDS
 G-28 SUCCEEDS
 G-23 SUCCEEDS
 G-31 : TRANSFORM G-12 TO DESIRED:OBJECT (FROM G-21 AND G-22)
 G-12 (PEG1 (DISK1 YES DISK2 YES) PEG2 (NIL) PEG3 (DISK3 YES DISK4 YES))
 G-32 : REDUCE UNDEF TO YES AT (PEG3 DISK2) OF G-12 (DIFFIC 205) (FROM G-31)
 G-12 (PEG1 (DISK1 YES DISK2 YES) PEG2 (NIL) PEG3 (DISK3 YES DISK4 YES))
 G-33 : APPLY MOVE:DISK TO G-12 (DIFFIC 105) (FROM G-32)
 ASSIGNS DISK - DISK2, TO:PEG - PEG3
 G-12 (PEG1 (DISK1 YES DISK2 YES) PEG2 (NIL) PEG3 (DISK3 YES DISK4 YES))
 G-34 : REDUCE UNDEF TO YES AT (PEG2 DISK1) OF G-12 (DIFFIC 105) (FROM G-33)
 G-12 (PEG1 (DISK1 YES DISK2 YES) PEG2 (NIL) PEG3 (DISK3 YES DISK4 YES))
 APPLY MOVE:DISK TO G-12 GET G-13 (FROM:PEG PEG1 TO:PEG PEG2 DISK DISK1)
 G-34 SUCCEEDS
 G-35 : APPLY MOVE:DISK TO G-13 (DIFFIC 105) (FROM G-33 AND G-34)
 ASSIGNS DISK - DISK2, TO:PEG - PEG3
 G-13 (PEG1 (DISK2 YES) PEG2 (DISK1 YES) PEG3 (DISK3 YES DISK4 YES))
 APPLY MOVE:DISK TO G-13 GET G-14 (FROM:PEG PEG1 TO:PEG PEG3 DISK DISK2)
 G-35 SUCCEEDS
 G-33 SUCCEEDS
 G-32 SUCCEEDS
 G-36 : TRANSFORM G-14 TO DESIRED:OBJECT (FROM G-31 AND G-32)
 G-14 (PEG1 NIL PEG2 (DISK1 YES) PEG3 (DISK2 YES DISK3 YES DISK4 YES))
 G-37 : REDUCE UNDEF TO YES AT (PEG3 DISK1) OF G-14 (DIFFIC 105) (FROM G-36)
 G-14 (PEG1 NIL PEG2 (DISK1 YES) PEG3 (DISK2 YES DISK3 YES DISK4 YES))
 APPLY MOVE:DISK TO G-14 GET G-15 (FROM:PEG PEG2 TO:PEG PEG3 DISK DISK1)
 G-15 IS THE SAME AS DESIRED:OBJECT
 G-37 SUCCEEDS
 G-38 : TRANSFORM DESIRED:OBJECT TO DESIRED:OBJECT (FROM G-36 AND G-37)
 DESIRED:OBJECT (PEG1 NIL PEG2 NIL PEG3 (DISK1 YES DISK2 YES DISK3 YES DISK4 YES))
 G-38 SUCCEEDS
 G-36 SUCCEEDS
 G-31 SUCCEEDS
 G-21 SUCCEEDS

G-1 SUCCEEDS

SUCCEEDED (TOP DESIRED OBJECT)

RUN TIME 28 MIN. 29.6 SEC

EXAM	TRY	FILE	MMCT	E/T	E/T	T/T
6364	3343	2615	0115	2.43	1.90	1.20
0.192	0.360	0.470	0.152	SEC AVG		

4523 INSPPTS 3532 DELETES 1105 MAPPINGS 463 NEW OBJECTS
 MAX SHPR LENGTH 113
 COPE (FILE FULL) (399) (2779) USED (17504 (1995)
 FIRED 112 OUT OF 275 PPODS

MISSIONARIES AND CANNIBALS FOR WITH ASSIGNMENT-ORDER NEW OBJ

G-1 : TRANSFORM INITIAL OBJECT TO DESIRED OBJECT (FROM TOP)
 LOC: PPOG LP-1 (LEFT CAN)
 LOC: PPOG LP-2 (LEFT MIS)
 LOC: PPOG LP-3 (LEFT BOAT)
 G-2 : REDUCE 3 TO 0 AT (LEFT CAN) OF INITIAL OBJECT (DIFFIC 203) (FROM G-1)
 APPLY CROSS-PIVER TO INITIAL OBJECT GET D-1 (FROM LEFT TO RIGHT NMIS 0 NEAN 2)
 G-2 SUCCEEDS
 G-3 : TRANSFORM D-1 TO DESIRED OBJECT (FROM G-1 AND G-2)
 D-1 (LEFT (CAN 1) MIS 3) RIGHT (BOAT YES CAN 2 MIS 0)
 G-4 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-1 (DIFFIC 203) (FROM G-3)
 D-1 (LEFT (CAN 1) MIS 3) RIGHT (BOAT YES CAN 2 MIS 0)
 G-5 : APPLY CROSS-PIVER TO D-1 (DIFFIC 105) (FROM G-4)
 ASSIGNS FROM - LEFT : NMIS - 1 : NMIS - 2
 D-1 (LEFT (CAN 1) MIS 3) RIGHT (BOAT YES CAN 2 MIS 0)
 G-6 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-1 (DIFFIC 105) (FROM G-5)
 D-1 (LEFT (CAN 1) MIS 3) RIGHT (BOAT YES CAN 2 MIS 0)
 APPLY CROSS-PIVER TO D-1 GET D-2 (FROM RIGHT TO LEFT NMIS 0 NEAN 1)
 G-6 SUCCEEDS
 G-7 : APPLY CROSS-PIVER TO D-2 (DIFFIC 105) (FROM G-5 AND G-6)
 ASSIGNS FROM - LEFT : NMIS - 1 : NMIS - 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 APPLY CROSS-PIVER TO D-2 GET D-3 (FROM LEFT TO RIGHT NMIS 1 NEAN 0)
 G-7 SUCCEEDS
 G-8 SUCCEEDS
 G-8 : TRANSFORM D-3 TO DESIRED OBJECT (FROM G-3 AND G-4)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-9 : REDUCE 2 TO 0 AT (LEFT CAN) OF D-3 (DIFFIC 202) (FROM G-8)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-10 : APPLY CROSS-PIVER TO D-3 (DIFFIC 105) (FROM G-9)
 ASSIGNS FROM - LEFT : NEAN - 1 : NEAN - 2
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-11 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-3 (DIFFIC 105) (FROM G-10)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 LOC: PPOG LP-4 (RIGHT MIS)
 APPLY CROSS-PIVER TO D-3 GET D-4 (FROM RIGHT TO LEFT NMIS 1 NEAN 0)
 D-4 IS THE SAME AS D-2
 G-11 SUCCEEDS
 G-12 : APPLY CROSS-PIVER TO D-2 (DIFFIC 105) (FROM G-10 AND G-11)
 ASSIGNS FROM - LEFT : NEAN - 1 : NEAN - 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 G-12 SUCCEEDS
 G-13 SUCCEEDS
 G-13 : TRANSFORM D-5 TO DESIRED OBJECT (FROM G-8 AND G-9)
 D-5 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
 G-14 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-5 (DIFFIC 203) (FROM G-13)
 D-5 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
 NO PROGRESS. G-14 FAILED
 TRANSFORM PETRY REJECT: G-13
 SELECT BY NEW OBJ. D-2
 G-15 : TRANSFORM D-2 TO DESIRED OBJECT (FROM G-11)
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 G-16 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-2 (DIFFIC 203) (FROM G-15)
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 APPLY CROSS-PIVER TO D-2 GET D-6 (FROM LEFT TO RIGHT NMIS 1 NEAN 0)
 D-6 IS THE SAME AS D-3

G-16 SUCCEEDS

G-17 : TRANSFORM D-3 TO DESIRED OBJECT (FROM G-15 AND G-16)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 REPEATED GOAL: G-17 G-8

PETRYING OLD G-11

APPLY CROSS-PIVER TO D-3 GET D-7 (FROM RIGHT TO LEFT NMIS 1 NEAN 1)
 D-7 IS THE SAME AS INITIAL OBJECT

G-11 SUCCEEDS

G-18 : APPLY CROSS-PIVER TO INITIAL OBJECT (DIFFIC 105) (FROM G-10
 AND G-11)

ASSIGNS FROM - LEFT : NEAN - 1 : NEAN - 2

APPLY CROSS-PIVER TO INITIAL OBJECT GET D-8 (FROM LEFT TO RIGHT NMIS
 0 NEAN 2)

D-8 IS THE SAME AS D-1

G-18 SUCCEEDS

G-10 SUCCEEDS

G-9 SUCCEEDS

G-19 : TRANSFORM D-1 TO DESIRED OBJECT (FROM G-8 AND G-9)

D-1 (LEFT (CAN 1) MIS 3) RIGHT (BOAT YES CAN 2 MIS 0)

REPEATED GOAL: G-19 G-3

PETRYING OLD G-11

G-20 : APPLY CROSS-PIVER TO D-3 (DIFFIC 201) (FROM G-11)

ASSIGNS TO - LEFT

D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))

NO PROGRESS. G-20 G-11 FAILED

G-20 FAILED

G-11 FAILED

PETRYING G-10

G-10 FAILED

PETRYING G-9

G-9 FAILED

TRANSFORM PETRY REJECT: G-8

PETRYING OLD G-6

APPLY CROSS-PIVER TO D-1 GET D-9 (FROM RIGHT TO LEFT NMIS 0 NEAN 2)

D-9 IS THE SAME AS INITIAL OBJECT

D-9 IS THE SAME AS INITIAL OBJECT

G-6 SUCCEEDS

G-21 : APPLY CROSS-PIVER TO INITIAL OBJECT (DIFFIC 105) (FROM G-5 AND
 G-6)

ASSIGNS FROM - LEFT : NMIS - 1 : NMIS - 2

APPLY CROSS-PIVER TO INITIAL OBJECT GET D-10 (FROM LEFT TO RIGHT NMIS 1
 NEAN 1)

D-10 IS THE SAME AS D-3

G-21 SUCCEEDS

G-5 SUCCEEDS

G-4 SUCCEEDS

G-22 : TRANSFORM D-3 TO DESIRED OBJECT (FROM G-3 AND G-4)

D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))

REPEATED GOAL: G-22 D-17

PETRYING OLD G-6

G-6 FAILED

PETRYING G-5

G-5 FAILED

PETRYING G-4

G-4 FAILED

TRANSFORM PETRY REJECT: G-3

PETRYING OLD G-14

G-23 : APPLY CROSS-PIVER TO D-5 (DIFFIC 105) (FROM G-14)

ASSIGNS FROM - LEFT : NMIS - 1 : NMIS - 2

D-5 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))

G-24 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-5 (DIFFIC 105) (FROM
 G-23)

D-5 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))

APPLY CROSS-PIVER TO D-5 GET D-11 (FROM RIGHT TO LEFT NMIS 0 NEAN 1)
 G-24 SUCCEEDS

G-25 : APPLY CROSS-PIVER TO D-11 (DIFFIC 105) (FROM G-23 AND G-24)
 ASSIGNS FROM - LEFT : NMIS - 1 : NMIS - 2

D-11 (LEFT (BOAT YES CAN 1 MIS 3) RIGHT (CAN 2 MIS 0))

APPLY CROSS-PIVER TO D-11 GET D-12 (FROM LEFT TO RIGHT NMIS 2 NEAN
 0)

G-25 SUCCEEDS

G-23 SUCCEEDS

G-14 SUCCEEDS

G-26 : TRANSFORM D-12 TO DESIRED OBJECT (FROM G-13 AND G-14)

D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))

G-27 : REDUCE 1 TO 0 AT (LEFT CAN) OF D-12 (DIFFIC 201) (FROM G-26)

D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))

G-28 : APPLY CROSS-PIVER TO D-12 (DIFFIC 105) (FROM G-27)

ASSIGNS FROM - LEFT, NCAN = 1
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 G-29 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-12 (DIFFIC 106) (FROM G-28)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 LOCIPPOG (P-5 (RIGHT CAN))
 APPLY CROSS:PIPER TO D-12 GET D-13 (FROM RIGHT TO LEFT MIS 1)
 NCAN 1)
 G-29 SUCCEEDS
 G-30 : APPLY CROSS:PIPER TO D-13 (DIFFIC 106) (FROM G-29 AND G-29)
 ASSIGNS FROM - LEFT, NCAN = 1
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 APPLY CROSS:PIPER TO D-13 GET D-14 (FROM LEFT TO RIGHT MIS 1)
 NCAN 1)
 D-14 IS THE SAME AS D-12
 G-30 SUCCEEDS
 G-28 SUCCEEDS
 G-27 SUCCEEDS
 G-31 : TRANSFORM D-12 TO DESIRED OBJECT (FROM G-26 AND G-27)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-31 G-26

SELECT BY NEW(OBJ, D-11)
 G-32 : TRANSFORM D-11 TO DESIRED OBJECT (FROM G-1)
 D-11 (LEFT (BOAT YES CAN 1 MIS 3) RIGHT (CAN 2 MIS 0))
 G-33 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-11 (DIFFIC 203) (FROM G-32)
 D-11 (LEFT (BOAT YES CAN 1 MIS 3) RIGHT (CAN 2 MIS 0))
 APPLY CROSS:PIPER TO D-11 GET D-15 (FROM LEFT TO RIGHT MIS 2 NCAN 0)
 D-15 IS THE SAME AS D-12
 G-33 SUCCEEDS
 G-34 : TRANSFORM D-12 TO DESIRED OBJECT (FROM G-32 AND G-33)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-34 G-26

SELECT BY NEW(OBJ, D-13)
 G-35 : TRANSFORM D-13 TO DESIRED OBJECT (FROM G-1)
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 G-36 : REDUCE 2 TO 0 AT (LEFT CAN) OF D-13 (DIFFIC 201) (FROM G-35)
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 APPLY CROSS:PIPER TO D-13 GET D-16 (FROM LEFT TO RIGHT MIS 1 NCAN 1)
 D-16 IS THE SAME AS D-12
 G-36 SUCCEEDS
 G-37 : TRANSFORM D-12 TO DESIRED OBJECT (FROM G-35 AND G-36)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-37 G-26

RETRYING OLD G-24
 APPLY CROSS:PIPER TO D-5 GET D-17 (FROM RIGHT TO LEFT MIS 0 NCAN 2)
 D-17 IS THE SAME AS D-2
 D-17 IS THE SAME AS D-2
 G-24 SUCCEEDS
 G-38 : APPLY CROSS:PIPER TO D-2 (DIFFIC 106) (FROM G-23 AND G-24)
 ASSIGNS FROM - LEFT, NMIS = 1, NMIS = 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 REPEATED GOAL: G-38 G-7

RETRYING OLD G-24
 G-24 FAILED
 RETRYING G-23
 G-23 FAILED
 RETRYING G-14
 G-14 FAILED
 TRANSFORM PETRY REJECT: G-13

RETRYING OLD G-29
 APPLY CROSS:PIPER TO D-17 GET D-18 (FROM RIGHT TO LEFT MIS 2 NCAN 0)
 D-18 IS THE SAME AS D-11
 G-29 SUCCEEDS
 G-39 : APPLY CROSS:PIPER TO D-11 (DIFFIC 105) (FROM G-28 AND G-29)
 ASSIGNS FROM - LEFT, NCAN = 1
 D-11 (LEFT (BOAT YES CAN 1 MIS 3) RIGHT (CAN 2 MIS 0))
 APPLY CROSS:PIPER TO D-11 GET D-19 (FROM LEFT TO RIGHT MIS 0 NCAN 1)
 D-19 IS THE SAME AS D-5
 G-39 SUCCEEDS
 G-28 SUCCEEDS
 G-27 SUCCEEDS
 G-40 : TRANSFORM D-5 TO DESIRED OBJECT (FROM G-26 AND G-27)
 D-5 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
 REPEATED GOAL: G-40 G-13

RETRYING OLD G-29

G-41 : APPLY CROSS:PIPER TO D-12 (DIFFIC 201) (FROM G-29)
 ASSIGNS TO - LEFT
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 NO PROGRESS, G-41 G-29 FAILED
 G-41 FAILED
 G-29 FAILED
 RETRYING G-28
 G-28 FAILED
 RETRYING G-27
 G-27 FAILED
 TRANSFORM PETRY REJECT: G-28

RETRYING OLD G-36
 G-42 : APPLY CROSS:PIPER TO D-13 (DIFFIC 201) (FROM G-36)
 ASSIGNS FROM - LEFT, NCAN = 1, NCAN = 2
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 G-43 : REDUCE 2 TO 1 AT (LEFT MIS) OF D-13 (DIFFIC 201) (FROM G-42)
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 APPLY CROSS:PIPER TO D-13 GET D-20 (FROM LEFT TO RIGHT MIS 1 NCAN 1)
 D-20 IS THE SAME AS D-12
 G-43 SUCCEEDS
 G-44 : APPLY CROSS:PIPER TO D-12 (DIFFIC 201) (FROM G-42 AND G-43)
 ASSIGNS FROM - LEFT, NCAN = 1, NCAN = 2
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 G-45 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-12 (DIFFIC 106) (FROM G-44)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-45 G-29

RETRYING OLD G-43
 G-46 : APPLY CROSS:PIPER TO D-13 (DIFFIC 201) (FROM G-43)
 ASSIGNS FROM - LEFT, NMIS = 1
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 G-47 : REDUCE 2 TO 1 AT (LEFT CAN) OF D-13 (DIFFIC 201) (FROM G-46)
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 APPLY CROSS:PIPER TO D-13 GET D-21 (FROM LEFT TO RIGHT MIS 1 NCAN 1)
 D-21 IS THE SAME AS D-12
 G-47 SUCCEEDS
 G-48 : APPLY CROSS:PIPER TO D-12 (DIFFIC 201) (FROM G-46 AND G-47)
 ASSIGNS FROM - LEFT, NMIS = 1
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 G-49 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-12 (DIFFIC 106) (FROM G-48)
 D-12 (LEFT (CAN 1 MIS 1) RIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-49 G-29

RETRYING OLD G-43
 G-43 FAILED
 RETRYING G-42
 G-42 FAILED
 RETRYING G-36
 G-50 : APPLY CROSS:PIPER TO D-13 (DIFFIC 201) (FROM G-36)
 ASSIGNS FROM - LEFT, NCAN = 1, NCAN = 2
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 G-51 : REDUCE 2 TO 0 AT (LEFT MIS) OF D-13 (DIFFIC 202) (FROM G-50)
 D-13 (LEFT (BOAT YES CAN 2 MIS 2) RIGHT (CAN 1 MIS 1))
 APPLY CROSS:PIPER TO D-13 GET D-22 (FROM LEFT TO RIGHT MIS 2 NCAN 0)
 G-51 SUCCEEDS
 G-52 : APPLY CROSS:PIPER TO D-22 (DIFFIC 202) (FROM G-50 AND G-51)
 ASSIGNS FROM - LEFT, NCAN = 1, NCAN = 2
 D-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))
 G-53 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-22 (DIFFIC 106) (FROM G-52)
 D-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))
 APPLY CROSS:PIPER TO D-22 GET D-23 (FROM RIGHT TO LEFT MIS 0 NCAN 1)
 G-53 SUCCEEDS
 G-54 : APPLY CROSS:PIPER TO D-23 (DIFFIC 106) (FROM G-52 AND G-53)
 ASSIGNS FROM - LEFT, NCAN = 1, NCAN = 2
 D-23 (LEFT (BOAT YES CAN 3 MIS 0) RIGHT (CAN 0 MIS 3))
 APPLY CROSS:PIPER TO D-23 GET D-24 (FROM LEFT TO RIGHT MIS 0 NCAN 2)
 G-54 SUCCEEDS
 G-52 SUCCEEDS
 G-50 SUCCEEDS
 G-36 SUCCEEDS
 G-55 : TRANSFORM D-24 TO DESIRED OBJECT (FROM G-35 AND G-36)
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 G-56 : REDUCE 1 TO 0 AT (LEFT CAN) OF D-24 (DIFFIC 201) (FROM G-55)
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 G-57 : APPLY CROSS:PIPER TO D-24 (DIFFIC 106) (FROM G-56)
 ASSIGNS FROM - LEFT, NCAN = 1
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 G-58 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-24 (DIFFIC 106) (FROM

G-57)
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 APPLY CROSS-PIVER TO D-24 GET D-25 (FROM RIGHT TO LEFT MIS 0 NEAN 1)
 G-58 SUCCEEDS
 G-59 : APPLY CROSS-PIVER TO D-25 (DIFFIC 105) (FROM G-57 AND G-58)
 ASSIGNS FROM - LEFT - NEAN - 1
 D-25 (LEFT (BOAT YES CAN 2 MIS 0) RIGHT (CAN 1 MIS 3))
 APPLY CROSS-PIVER TO D-25 GET D-26 (FROM LEFT TO RIGHT MIS 0 NEAN 1)
 D-26 IS THE SAME AS D-24
 G-59 SUCCEEDS
 G-57 SUCCEEDS
 G-56 SUCCEEDS
 G-60 : TRANSFORM D-24 TO DESIRED OBJECT (FROM G-55 AND G-56)
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 REPEATED GOAL: G-60 G-55
 SELECT BY NEW OBJ. D-22
 G-61 : TRANSFORM D-22 TO DESIRED OBJECT (FROM G-1)
 D-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))
 G-62 : REDUCE 2 TO 0 AT (LEFT CAN) OF D-22 (DIFFIC 202) (FROM G-61)
 D-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))
 G-63 : APPLY CROSS-PIVER TO D-22 (DIFFIC 105) (FROM G-62)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 D-22 (LEFT (CAN 2 MIS 0) RIGHT (BOAT YES CAN 1 MIS 3))
 REPEATED GOAL: G-63 G-52
 SELECT BY NEW OBJ. D-23
 G-64 : TRANSFORM D-23 TO DESIRED OBJECT (FROM G-1)
 D-23 (LEFT (BOAT YES CAN 3 MIS 0) RIGHT (CAN 0 MIS 3))
 G-65 : REDUCE 3 TO 0 AT (LEFT CAN) OF D-23 (DIFFIC 203) (FROM G-64)
 D-23 (LEFT (BOAT YES CAN 3 MIS 0) RIGHT (CAN 0 MIS 3))
 APPLY CROSS-PIVER TO D-23 GET D-27 (FROM LEFT TO RIGHT MIS 0 NEAN 2)
 D-27 IS THE SAME AS D-24
 G-65 SUCCEEDS
 G-66 : TRANSFORM D-24 TO DESIRED OBJECT (FROM G-64 AND G-65)
 D-24 (LEFT (CAN 1 MIS 0) RIGHT (BOAT YES CAN 2 MIS 3))
 REPEATED GOAL: G-66 G-55
 SELECT BY NEW OBJ. D-25
 G-67 : TRANSFORM D-25 TO DESIRED OBJECT (FROM G-1)
 D-25 (LEFT (BOAT YES CAN 2 MIS 0) RIGHT (CAN 1 MIS 3))
 G-68 : REDUCE 2 TO 0 AT (LEFT CAN) OF D-25 (DIFFIC 202) (FROM G-67)
 D-25 (LEFT (BOAT YES CAN 2 MIS 0) RIGHT (CAN 1 MIS 3))
 APPLY CROSS-PIVER TO D-25 GET D-28 (FROM LEFT TO RIGHT MIS 0 NEAN 2)
 D-28 IS THE SAME AS DESIRED OBJECT
 G-68 SUCCEEDS
 G-69 : TRANSFORM DESIRED OBJECT TO DESIRED OBJECT (FROM G-67 AND G-68)
 DESIRED OBJECT (LEFT (CAN 0 MIS 0) RIGHT (BOAT YES CAN 3 MIS 3))
 G-69 SUCCEEDS
 G-67 SUCCEEDS
 G-1 SUCCEEDS

SUCCEED (TOP DESIRED OBJECT)

RUN TIME 43 MIN. 37.6 SEC

EXAM	TRY	FILE	WACT	E/T	E/T	T/T
11329	6461	4313	1404	2.63	1.75	1.50
0.231	0.405	0.647	0.106	SEC AVG		

7922 INSERTS 6119 DELETES 1755 WARNINGS 675 NEW OBJECTS
 MAX (SMPX LENGTH 125
 COPE (FREE, FULL): (15400, 3650) USED (14610, 261)

MISSIONARIES AND CANNIBALS (C) WITH ASSIGNMENT ODDP ONLY

G-1 : TRANSFORM INITIAL OBJECT TO DESIRED OBJECT (FROM TOP)
 LOC:PROG LP-1 (LEFT CAN)
 LOC:PROG LP-2 (LEFT MIS)
 LOC:PROG LP-3 (LEFT RIGHT)
 G-2 : REDUCE 3 TO 0 AT (LEFT CAN) OF INITIAL OBJECT (DIFFIC 203) (FROM G-1)
 APPLY CROSS-PIVER TO INITIAL OBJECT GET D-1 (FROM LEFT TO RIGHT MIS 1 NEAN 2)
 G-2 SUCCEEDS
 G-3 : TRANSFORM D-1 TO DESIRED OBJECT (FROM G-1 AND G-2)
 D-1 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 2 MIS 0))
 G-4 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-1 (DIFFIC 203) (FROM G-3)
 D-1 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 2 MIS 0))

G-5 : APPLY CROSS-PIVER TO D-1 (DIFFIC 105) (FROM G-4)
 ASSIGNS FROM - LEFT - MIS - 1 - MIS - 2
 D-1 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 2 MIS 0))
 G-6 : REDUCE 1 TO 0 AT (LEFT BOAT) OF D-1 (DIFFIC 105) (FROM G-5)
 D-1 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 2 MIS 0))
 APPLY CROSS-PIVER TO D-1 GET D-2 (FROM RIGHT TO LEFT MIS 0 NEAN 1)
 G-6 SUCCEEDS
 G-7 : APPLY CROSS-PIVER TO D-2 (DIFFIC 105) (FROM G-5 AND G-6)
 ASSIGNS FROM - LEFT - MIS - 1 - MIS - 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 APPLY CROSS-PIVER TO D-2 GET D-3 (FROM LEFT TO RIGHT MIS 1 NEAN 0)
 G-7 SUCCEEDS
 G-5 SUCCEEDS
 G-4 SUCCEEDS
 G-8 : TRANSFORM D-3 TO DESIRED OBJECT (FROM G-3 AND G-4)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-9 : REDUCE 2 TO 0 AT (LEFT CAN) OF D-3 (DIFFIC 202) (FROM G-8)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 LOC:PROG LP-4 (RIGHT MIS)
 G-10 : APPLY CROSS-PIVER TO D-3 (DIFFIC 105) (FROM G-9)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-11 : REDUCE 1 TO 0 AT (LEFT BOAT) OF D-3 (DIFFIC 105) (FROM G-10)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 APPLY CROSS-PIVER TO D-3 GET D-4 (FROM RIGHT TO LEFT MIS 1 NEAN 0)
 D-4 IS THE SAME AS D-2
 G-11 SUCCEEDS
 G-12 : APPLY CROSS-PIVER TO D-2 (DIFFIC 105) (FROM G-10 AND G-11)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 APPLY CROSS-PIVER TO D-2 GET D-5 (FROM LEFT TO RIGHT MIS 0 NEAN 2)
 G-12 SUCCEEDS
 G-10 SUCCEEDS
 G-9 SUCCEEDS
 G-13 : TRANSFORM D-5 TO DESIRED OBJECT (FROM G-8 AND G-9)
 D-5 (LEFT (CAN 0 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
 G-14 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-5 (DIFFIC 203) (FROM G-13)
 D-5 (LEFT (CAN 0 MIS 3) RIGHT (BOAT YES CAN 3 MIS 0))
 NO PROGRESS. G-14 FAILED
 TRANSFORM PETPY REJECT: G-13
 RETRYING OLD G-11
 APPLY CROSS-PIVER TO D-3 GET D-6 (FROM RIGHT TO LEFT MIS 1 NEAN 1)
 D-6 IS THE SAME AS INITIAL OBJECT
 G-11 SUCCEEDS
 G-15 : APPLY CROSS-PIVER TO INITIAL OBJECT (DIFFIC 105) (FROM G-11
 AND G-11)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 APPLY CROSS-PIVER TO INITIAL OBJECT GET D-7 (FROM LEFT TO RIGHT MIS
 0 NEAN 2)
 D-7 IS THE SAME AS D-1
 G-15 SUCCEEDS
 G-10 SUCCEEDS
 G-9 SUCCEEDS
 G-16 : TRANSFORM D-1 TO DESIRED OBJECT (FROM G-8 AND G-9)
 D-1 (LEFT (CAN 1 MIS 3) RIGHT (BOAT YES CAN 2 MIS 0))
 REPEATED GOAL: G-16 G-3

RETRYING OLD G-11
 G-17 : APPLY CROSS-PIVER TO D-3 (DIFFIC 201) (FROM G-11)
 ASSIGNS TO - LEFT
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 NO PROGRESS. G-17 G-11 FAILED
 G-17 FAILED
 G-11 FAILED
 RETRYING G-10
 G-10 FAILED
 RETRYING G-9
 G-10 : APPLY CROSS-PIVER TO D-3 (DIFFIC 201) (FROM G-9)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 G-19 : REDUCE 1 TO 0 AT (RIGHT MIS) OF D-3 (DIFFIC 201) (FROM G-10)
 D-3 (LEFT (CAN 2 MIS 2) RIGHT (BOAT YES CAN 1 MIS 1))
 APPLY CROSS-PIVER TO D-3 GET D-8 (FROM RIGHT TO LEFT MIS 1 NEAN 0)
 D-8 IS THE SAME AS D-2
 D-8 IS THE SAME AS D-2
 G-19 SUCCEEDS
 G-20 : APPLY CROSS-PIVER TO D-2 (DIFFIC 201) (FROM G-10 AND G-19)
 ASSIGNS FROM - LEFT - NEAN - 1 - NEAN - 2
 D-2 (LEFT (BOAT YES CAN 2 MIS 3) RIGHT (CAN 1 MIS 0))
 REPEATED GOAL: G-20 G-12

PETRYING OLD G-6
 APPLY CROSS-PIPER TO D-1 GET D-9 (FROM RIGHT TO LEFT MISS 0 NEAN 2)
 D-9 IS THE SAME AS INITIAL OBJECT
 D-9 IS THE SAME AS INITIAL OBJECT
 G-6 SUCCEEDS
 G-21 : APPLY CROSS-PIPER TO INITIAL OBJECT (DIFFIC 105) (FROM G-5 AND G-6)
 ASSIGNS FROM - LEFT , MISS = 1 , MISS = 2
 APPLY CROSS-PIPER TO INITIAL OBJECT GET D-10 (FROM LEFT TO RIGHT MISS 1 NEAN 1)
 D-10 IS THE SAME AS D-3
 G-21 SUCCEEDS
 G-5 SUCCEEDS
 G-4 SUCCEEDS
 G-22 : TRANSFORM D-3 TO DESIRED OBJECT (FROM G-3 AND G-4)
 D-3 (LEFT (CAN 2 HIS 2) RIGHT (BOAT YES CAN 1 HIS 1))
 REPEATED GOAL: G-22 G-8

PETRYING OLD G-6
 G-6 FAILED
 PETRYING G-5
 G-5 FAILED
 PETRYING G-4
 G-4 FAILED
 TRANSFORM PETRY REJECT: G-3

PETRYING OLD G-19
 APPLY CROSS-PIPER TO D-3 GET D-11 (FROM RIGHT TO LEFT MISS 1 NEAN 1)
 D-11 IS THE SAME AS INITIAL OBJECT
 D-11 IS THE SAME AS INITIAL OBJECT
 G-19 SUCCEEDS
 G-23 : APPLY CROSS-PIPER TO INITIAL OBJECT (DIFFIC 201) (FROM G-10 AND G-19)
 ASSIGNS FROM - LEFT , NEAN = 1 , NEAN = 2
 REPEATED GOAL: G-23 G-15
 PETRYING OLD G-19
 G-19 FAILED
 PETRYING G-18
 G-18 FAILED
 PETRYING G-9
 G-9 FAILED
 TRANSFORM PETRY REJECT: G-8

PETRYING OLD G-14
 G-24 : APPLY CROSS-PIPER TO D-5 (DIFFIC 105) (FROM G-14)
 ASSIGNS FROM - LEFT , MISS = 1 , MISS = 2
 D-5 (LEFT (CAN 0 HIS 3) RIGHT (BOAT YES CAN 3 HIS 0))
 G-25 : APPLY CROSS-PIPER TO D-5 (DIFFIC 105) (FROM G-24)
 D-5 (LEFT (CAN 0 HIS 3) RIGHT (BOAT YES CAN 3 HIS 0))
 G-25 SUCCEEDS
 G-26 : APPLY CROSS-PIPER TO D-12 (DIFFIC 105) (FROM G-24 AND G-25)
 ASSIGNS FROM - LEFT , MISS = 1 , MISS = 2
 D-12 (LEFT (BOAT YES CAN 1 HIS 3) RIGHT (CAN 2 HIS 0))
 APPLY CROSS-PIPER TO D-12 GET D-13 (FROM LEFT TO RIGHT MISS 2 NEAN 0)
 G-26 SUCCEEDS
 G-24 SUCCEEDS
 G-27 : TRANSFORM D-13 TO DESIRED OBJECT (FROM G-13 AND G-14)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 G-28 : REDUCE 1 TO 0 AT (LEFT CAN) OF D-13 (DIFFIC 201) (FROM G-27)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 G-29 : APPLY CROSS-PIPER TO D-13 (DIFFIC 105) (FROM G-28)
 ASSIGNS FROM - LEFT , NEAN = 1
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 G-30 : REDUCE UNOFF TO YES AT (LEFT BOAT) OF D-13 (DIFFIC 105) (FROM G-29)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 LOC PROG LP-5 (PIGHT CAN)
 APPLY CROSS-PIPER TO D-13 GET D-14 (FROM RIGHT TO LEFT MISS 1 NEAN 1)
 G-30 SUCCEEDS
 G-31 : APPLY CROSS-PIPER TO D-14 (DIFFIC 105) (FROM G-29 AND G-30)
 ASSIGNS FROM - LEFT , NEAN = 1
 D-14 (LEFT (BOAT YES CAN 2 HIS 2) RIGHT (CAN 1 HIS 1))
 APPLY CROSS-PIPER TO D-14 GET D-15 (FROM LEFT TO RIGHT MISS 1 NEAN 1)
 D-15 IS THE SAME AS D-13
 G-31 SUCCEEDS
 G-29 SUCCEEDS

G-20 SUCCEEDS
 G-32 : TRANSFORM D-13 TO DESIRED OBJECT (FROM G-27 AND G-20)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 REPEATED GOAL: G-32 G-27

PETRYING OLD G-25
 APPLY CROSS-PIPER TO D-5 GET D-16 (FROM RIGHT TO LEFT MISS 0 NEAN 2)
 D-16 IS THE SAME AS D-2
 D-16 IS THE SAME AS D-2
 G-25 SUCCEEDS
 G-23 : APPLY CROSS-PIPER TO D-2 (DIFFIC 105) (FROM G-24 AND G-25)
 ASSIGNS FROM - LEFT , MISS = 1 , MISS = 2
 D-2 (LEFT (BOAT YES CAN 2 HIS 3) RIGHT (CAN 1 HIS 0))
 REPEATED GOAL: G-23 G-7

PETRYING OLD G-25
 G-25 FAILED
 PETRYING G-24
 G-24 FAILED
 PETRYING G-14
 G-14 FAILED
 TRANSFORM PETRY REJECT: G-13

PETRYING OLD G-30
 APPLY CROSS-PIPER TO D-13 GET D-17 (FROM RIGHT TO LEFT MISS 2 NEAN 0)
 D-17 IS THE SAME AS D-12
 G-30 SUCCEEDS
 G-34 : APPLY CROSS-PIPER TO D-12 (DIFFIC 105) (FROM G-29 AND G-30)
 ASSIGNS FROM - LEFT , NEAN = 1
 D-12 (LEFT (BOAT YES CAN 1 HIS 3) RIGHT (CAN 2 HIS 0))
 APPLY CROSS-PIPER TO D-12 GET D-18 (FROM LEFT TO RIGHT MISS 0 NEAN 1)
 D-18 IS THE SAME AS D-5
 G-34 SUCCEEDS
 G-29 SUCCEEDS
 G-20 SUCCEEDS
 G-35 : TRANSFORM D-5 TO DESIRED OBJECT (FROM G-27 AND G-20)
 D-5 (LEFT (CAN 0 HIS 3) RIGHT (BOAT YES CAN 3 HIS 0))
 REPEATED GOAL: G-35 G-13

PETRYING OLD G-30
 G-36 : APPLY CROSS-PIPER TO D-13 (DIFFIC 201) (FROM G-30)
 ASSIGNS TO - LEFT
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 NO PROGRESS. G-36 G-30 FAILED
 G-36 FAILED
 G-30 FAILED
 PETRYING G-29
 G-29 FAILED
 PETRYING G-20
 G-37 : APPLY CROSS-PIPER TO D-13 (DIFFIC 201) (FROM G-20)
 ASSIGNS FROM - LEFT , NEAN = 1
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 G-38 : REDUCE 2 TO 1 AT (RIGHT HIS) OF D-13 (DIFFIC 201) (FROM G-37)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 APPLY CROSS-PIPER TO D-13 GET D-19 (FROM RIGHT TO LEFT MISS 1 NEAN 1)
 D-19 IS THE SAME AS D-14
 D-19 IS THE SAME AS D-14
 G-38 SUCCEEDS
 G-39 : APPLY CROSS-PIPER TO D-14 (DIFFIC 201) (FROM G-37 AND G-38)
 ASSIGNS FROM - LEFT , NEAN = 1
 D-14 (LEFT (BOAT YES CAN 2 HIS 2) RIGHT (CAN 1 HIS 1))
 REPEATED GOAL: G-39 G-31

PETRYING OLD G-20
 G-20 FAILED
 TRANSFORM PETRY REJECT: G-27

PETRYING OLD G-20
 G-40 : APPLY CROSS-PIPER TO D-13 (DIFFIC 201) (FROM G-30)
 ASSIGNS FROM - RIGHT , MISS = 1
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 G-41 : REDUCE 2 TO 1 AT (RIGHT CAN) OF D-13 (DIFFIC 201) (FROM G-40)
 D-13 (LEFT (CAN 1 HIS 1) RIGHT (BOAT YES CAN 2 HIS 2))
 APPLY CROSS-PIPER TO D-13 GET D-20 (FROM RIGHT TO LEFT MISS 1 NEAN 1)
 D-20 IS THE SAME AS D-14
 D-20 IS THE SAME AS D-14
 G-41 SUCCEEDS

G-42 : APPLY CROSS-PIPER TO D-14 (DIFFIC 201) (FROM G-40 AND G-41)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 LOC: PROG LP-6 (PIGHT ADAT)
 G-43 : REDUCE (NOFF TO YES AT (PIGHT BOAT) OF D-14 (DIFFIC 195) (FROM G-42)
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 APPLY CROSS-PIPER TO D-14 GET D-21 (FROM LEFT TO PIGHT MHS 1) (CAN 1)
 D-21 IS THE SAME AS D-14
 G-43 SUCCEEDS
 G-44 : APPLY CROSS-PIPER TO D-13 (DIFFIC 195) (FROM G-42 AND G-43)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-13 (LEFT (CAN 1 MIS 1) PIGHT (BOAT YES CAN 2 MIS 2))
 APPLY CROSS-PIPER TO D-13 GET D-22 (FROM PIGHT TO LEFT MHS 1) (CAN 1)
 D-22 IS THE SAME AS D-14
 D-22 IS THE SAME AS D-14
 G-44 SUCCEEDS
 G-42 SUCCEEDS
 G-40 SUCCEEDS
 G-30 SUCCEEDS
 G-45 : APPLY CROSS-PIPER TO D-14 (DIFFIC 201) (FROM G-37 AND G-30)
 ASSIGNS FROM - LEFT - NCAN - 1
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 REPEATED GOAL: G-45 G-31
 RETRYING OLD G-43
 APPLY CROSS-PIPER TO D-14 GET D-23 (FROM LEFT TO PIGHT MHS 2) (CAN 1)
 G-43 SUCCEEDS
 G-46 : APPLY CROSS-PIPER TO D-23 (DIFFIC 195) (FROM G-42 AND G-43)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 G-47 : REDUCE 3 TO 2 AT (PIGHT MIS) OF D-23 (DIFFIC 201) (FROM G-46)
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 NO PROGRESS. G-47 FAILED
 RETRYING G-46
 G-48 : REDUCE 3 TO 1 AT (PIGHT MIS) OF D-23 (DIFFIC 202) (FROM G-46)
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 NO PROGRESS. G-48 FAILED
 RETRYING G-46
 G-46 FAILED
 RETRYING G-43
 G-49 : APPLY CROSS-PIPER TO D-14 (DIFFIC 201) (FROM G-43)
 ASSIGNS TO - PIGHT
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 NO PROGRESS. G-49 G-43 FAILED
 G-49 FAILED
 G-43 FAILED
 RETRYING G-47
 G-50 : REDUCE 2 TO 1 AT (LEFT CAN) OF D-14 (DIFFIC 201) (FROM G-42)
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 NO PROGRESS. G-50 FAILED
 RETRYING G-42
 G-42 FAILED
 RETRYING G-41
 G-51 : APPLY CROSS-PIPER TO D-13 (DIFFIC 201) (FROM G-41)
 ASSIGNS FROM - PIGHT - NCAN - 1
 D-13 (LEFT (CAN 1 MIS 1) PIGHT (BOAT YES CAN 2 MIS 2))
 G-52 : REDUCE 2 TO 1 AT (PIGHT MIS) OF D-13 (DIFFIC 201) (FROM G-51)
 D-13 (LEFT (CAN 1 MIS 1) PIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-52 G-30
 RETRYING OLD G-30
 G-30 FAILED
 RETRYING G-37
 G-37 FAILED
 REPEATED FAIL: G-20
 RETRYING OLD G-41
 G-41 FAILED
 RETRYING G-40
 G-40 FAILED
 REPEATED FAIL: G-30
 RETRYING OLD G-2
 APPLY CROSS-PIPER TO INITIAL OBJECT GET D-26 (FROM LEFT TO RIGHT MHS 1) (CAN 1)
 D-26 IS THE SAME AS D-3
 G-2 SUCCEEDS
 G-62 : TRANSFER D-3 TO DESIRED OBJECT (FROM G-1 AND G-2)
 D-3 (LEFT (CAN 2 MIS 2) PIGHT (BOAT YES CAN 1 MIS 1))
 REPEATED GOAL: G-62 G-22
 RETRYING OLD G-47
 G-53 : APPLY CROSS-PIPER TO D-23 (DIFFIC 201) (FROM G-47)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 REPEATED GOAL: G-53 G-46
 RETRYING OLD G-47
 G-54 : APPLY CROSS-PIPER TO D-23 (DIFFIC 202) (FROM G-47)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 NO PROGRESS. G-54 G-47 FAILED
 G-54 FAILED
 G-47 FAILED
 REPEATED FAIL: G-46
 RETRYING OLD G-50
 APPLY CROSS-PIPER TO D-14 GET D-24 (FROM LEFT TO RIGHT MHS 1) (CAN 1)
 D-24 IS THE SAME AS D-13
 G-50 SUCCEEDS
 G-55 : APPLY CROSS-PIPER TO D-13 (DIFFIC 201) (FROM G-42 AND G-50)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-13 (LEFT (CAN 1 MIS 1) PIGHT (BOAT YES CAN 2 MIS 2))
 REPEATED GOAL: G-55 G-44
 RETRYING OLD G-50
 G-56 : APPLY CROSS-PIPER TO D-14 (DIFFIC 201) (FROM G-50)
 ASSIGNS FROM - LEFT - NCAN - 1
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 REPEATED GOAL: G-56 G-31
 RETRYING OLD G-50
 G-50 FAILED
 REPEATED FAIL: G-42
 RETRYING OLD G-40
 APPLY CROSS-PIPER TO D-23 GET D-25 (FROM RIGHT TO LEFT MHS 2) (CAN 1)
 D-25 IS THE SAME AS D-14
 D-25 IS THE SAME AS D-14
 G-40 SUCCEEDS
 G-57 : APPLY CROSS-PIPER TO D-14 (DIFFIC 202) (FROM G-40 AND G-48)
 ASSIGNS FROM - PIGHT - MHS - 1
 D-14 (LEFT (BOAT YES CAN 2 MIS 2) PIGHT (CAN 1 MIS 1))
 REPEATED GOAL: G-57 G-42
 RETRYING OLD G-40
 G-58 : APPLY CROSS-PIPER TO D-23 (DIFFIC 201) (FROM G-48)
 ASSIGNS FROM - PIGHT - MHS - 1 - MHS - 2
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 G-59 : REDUCE 3 TO 2 AT (PIGHT MIS) OF D-23 (DIFFIC 201) (FROM G-58)
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 REPEATED GOAL: G-59 G-47
 RETRYING OLD G-40
 G-60 : APPLY CROSS-PIPER TO D-23 (DIFFIC 202) (FROM G-48)
 ASSIGNS FROM - PIGHT - MHS - 1 - MHS - 2
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 G-61 : REDUCE 3 TO 1 AT (PIGHT MIS) OF D-23 (DIFFIC 202) (FROM G-60)
 D-23 (LEFT (CAN 2 MIS 1) PIGHT (BOAT YES CAN 1 MIS 3))
 REPEATED GOAL: G-61 G-48
 RETRYING OLD G-40
 G-40 FAILED
 REPEATED FAIL: G-48

RETRYING OLD G-2
 APPLY CROSS-PIPER TO INITIAL OBJECT GET D-27 (FROM LEFT TO RIGHT MIS & NEAR 1)
 G-2 SUCCEEDS
 G-63 : TRANSFORM D-27 TO DESIRED OBJECT (FROM G-1 AND G-2)
 D-27 (LEFT (CAN 2 MIS 3) RIGHT (BOAT YES CAN 1 MIS 0))
 G-64 : REDUCE 3 TO 0 AT (LEFT MIS) OF D-27 (DIFFIC 203) (FROM G-63)
 D-27 (LEFT (CAN 2 MIS 3) RIGHT (BOAT YES CAN 1 MIS 0))
 G-65 : APPLY CROSS-PIPER TO D-27 (DIFFIC 105) (FROM G-64)
 ASSIGNS FROM : LEFT : MIS = 1 : MIS = 2
 D-27 (LEFT (CAN 2 MIS 3) RIGHT (BOAT YES CAN 1 MIS 0))
 G-66 : REDUCE UNDER TO YES AT (LEFT BOAT) OF D-27 (DIFFIC 105) (FROM G-65)
 D-27 (LEFT (CAN 2 MIS 3) RIGHT (BOAT YES CAN 1 MIS 0))
 APPLY CROSS-PIPER TO D-27 GET D-28 (FROM RIGHT TO LEFT MIS & NEAR 1)
 D-28 IS THE SAME AS INITIAL OBJECT
 D-28 IS THE SAME AS INITIAL OBJECT
 G-66 SUCCEEDS
 G-67 : APPLY CROSS-PIPER TO INITIAL OBJECT (DIFFIC 105) (FROM G-65 AND G-66)
 ASSIGNS FROM : LEFT : MIS = 1 : MIS = 2
 REPEATED COM : G-67 G-2)

RETRYING OLD G-66
 G-66 FAILED
 RETRYING G-65
 G-65 FAILED
 RETRYING G-64
 G-64 FAILED
 TRANSFORM PETPY PROJECT: G-63

RETRYING OLD G-2
 G-2 FAILED
 TRANSFORM PETPY PROJECT: G-1

RUN TIME 42 MIN. 56 3 SEC

EXAM	TRY	FILE	IMPACT	E/T	E/T	T/T
11426	6746	4700	14090	2.66	1.69	1.57
0.225	0.382	0.601	0.183	SEC AVG		

7969 INSERTS 6121 DELETES 1966 WARNINGS 634 NEW OBJECTS
 MAX : SMPX LENGTH 142
 CORE (FREE FULL): (12934 : 3101) USED (-5500 : -2101)
 FIRED 140 OUT OF 261 PPODS

DATE
FILME
- 8